DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/89D-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering; 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, Ohio 45433-6533	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AU CADRE/WG; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) AU CADRE/WG Maxwell AFB, Al 36112-5532	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
A GRAPHICAL PLAYER INTERFACE TO THE THEATER WAR EXERCISE (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Peter J. Gordon, Captain USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	891206	60

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	User Interface, Interactive Graphics, Wargames
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

(see reverse)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION	
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

AFIT/GCS/ENG/89-

Abstract

The Theater War Exercise (TWX) is a two sided, theater level, computer assisted, air power employment decision making exercise. The primary purpose of the exercise is to give senior military officers a feel for the factors involved with high level decision-making in an air/land conventional warfare situation.

This thesis investigates the feasibility of replacing the recently developed graphical output display and the seperate form-based input interface with one combined interface. The result allows the players to make their inputs directly through the graphics display. The new interface is easy to use and learn, reducing distractions from the primary purpose of the exercise.

The interface prototype was developed using a combined methodology that included features from the Classic Life Cycle Methodology, Iterative Design Methodology, and Prototyping. The prototype was designed to fit within the framework of the output display, incorporating the full functionality of the input interface while not degrading the functionality of the output display. Full system installation was delayed until the ORACLE database was loaded onto the SUN workstations.

Additional benefits derived from this effort were insights into computer interaction and human factors, and the evaluation of graphic toolkits.

AFIT/GCS/ENG/89D-5

A Graphical Player Interface To The Theater War Exercise

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science, Computer Systems

Peter J Gordon, B.S.
Captain, USAF

December, 1989

AFIT/GCS/ENG/89D-5

A Graphical Player Interface To The Theater War Exercise

THESIS

Peter J Gordon
Captain, USAF

AFIT/GCS/ENG/89D-5

Preface

The goal of this thesis was to investigate the feasibility of replacing a recently developed graphical output display and separate form-based input interface of a theater-level wargaming exercise (TWX) with one combined interface. The result allows the players to make their inputs directly through the graphics display. The new interface is be easy to use and learn, reducing distractions from the primary purpose of the exercise.

This thesis presents an introduction to the TWX system and the proposed modifications, reviews current literature related to the design of graphic interfaces, describes and compares different graphic software systems available, and describes the methodology and the steps used in developing the system. Finally the finished prototype is examined in detail.

I am grateful for the support provided by my thesis advisor, Major Mark A. Roth, and by my thesis committee members, Lieutenant Colonel James N. Robinson and Major Phil Amburn. In addition, I would like to express my gratitude to my fellow students for their support. Finally, and most importantly, I would like to thank my wife, Patrice, my sons, Donald and Peter, and my daughter, Sara, for their emotional support and love.

Peter J Gordon

Table of Contents

Page

Preface ii

Table of Contents iii

List of Figures vi

Abstract vii

I. Introduction 1

1.1 Background 1

1.2 Problem 3

1.3 Assumptions 4

1.4 Scope 4

1.5 Approach/Methodology 5

1.6 Sequence of Presentation 6

II. Review of Literature 7

2.1 Computer Interfaces 7

2.2 Human Factors and Interface Design 7

2.3 Icons and Direct Interface Manipulation 9

2.4 Menu Design and Implementation 10

III. Graphics Software Toolkits 12

3.1 Background 12

3.2 Graphics Standards 13

3.3 Core, GKS, and PHIGS 14

3.3.1 Core. 14

Page

3.3.2 GKS. 15

3.3.3 PHIGS. 16

3.4 HOOPS 17

3.5 X Window System 19

IV. Design Methodology 23

4.1 Classic Life Cycle. 23

4.2 Prototyping. 24

4.3 Iterative Design. 26

4.4 The Methodology Decision. 26

V. Requirements Analysis 29

5.1 Given Requirements 29

5.2 Human Factors 30

5.3 AAFCE Input Requirements 30

5.3.1 Report Functions. 31

5.3.2 Input Functions. 31

5.3.3 Introductory Functions. 33

5.4 Output Display Interface 33

VI. System Design 38

6.1 Conception 38

6.2 Graphical Display 38

6.2.1 Menus. 40

6.2 2 Buttons. 43

6.2.3 Display Forms. 44

6.3 User Input and Input Devices 44

Page

VII. Description of the Prototype 47

7.1 Configuration 47

7.2 Sign-on Procedures 47

7.3 Modifications to the Output Display Interface 48

7.3.1 The Display Keyboard. 48

7.3.2 Help Function 49

7.3.3 Show_Detail Function. 49

7.3.4 Unit Symbols. 49

7.4 New Extensions 51

7.4.1 AAFCE Menu 51

7.4.2 Display Functions 51

7.4.3 Input Functions. 52

VIII. Conclusion and Recommendations 54

8.1 Review of Steps 54

8.2 Strength sand Weaknesses of the New System 55

8.3 Enhancements 56

8.4 Conclusion 57

Bibliography 58

Vita 60

List of Figures

Figure		Page
1	The Classic Life Cycle	23
2.	Prototyping	25
3.	Evolutionary Development	26
4.	The Combined Methodology	28
5.	Structure Chart of the Form-Based Input Interface	32
6.	Structure Chart of the Output Display Interface - Part1	34
7.	Structure Chart of the Output Display Interface - Part2	35
8.	Structure Chart of the Output Display Interface - Part 3	36
9.	HOOPS Object Hierarchy	37
10.	Conceptual Model of the Display and Interface	39
11.	Overlapping Menus with Highlighting Options	42
12.	The Display Keyboard	49
13.	Intelligence Level	50
14.	Base and Unit Symbols	50

AFIT/GCS/ENG/89D-5

Abstract

The Theater War Exercise (TWX) is a two sided, theater level, computer assisted, air power employment decision making exercise. The primary purpose of the exercise is to give senior military officers a feel for the factors involved with high level decision-making in an air/land conventional warfare situation.

This thesis investigates the feasibility of replacing the recently developed graphical output display and the seperate form-based input interface with one combined interface. The result allows the players to make their inputs directly through the graphics display. The new interface is easy to use and learn, reducing distractions from the primary purpose of the exercise.

The interface prototype was developed using a combined methodology that included features from the Classic Life Cycle Methodology, Iterative Design Methodology, and Prototyping. The prototype was designed to fit within the framework of the output display, incorporating the full functionality of the input interface while not degrading the functionality of the output display. Full system installation was delayed until the ORACLE database was loaded onto the SUN workstations.

Additional benefits derived from this effort were insights into computer interaction and human factors, and the evaluation of graphic toolkits.

A Graphical Player Interface To The Theater War Exercise

I Introduction

The Theater War Exercise (TWX) is a two sided, theater level, computer assisted, air power employment decision making exercise. The exercise is conducted by the Air Force Wargaming Center (AFWC) as part of the Air War College curriculum. The primary purpose of the exercise is to give senior military officers a feel for the factors involved with high level decision-making in an air/land conventional warfare situation.

The decisions required are typical of those an air component commander and staff would make. The decisions are fed into the TWX air and land battle programs which simulate the use of airpower strategy, doctrine, and warfighting principles. The players receive the computerized battle results, orders of battle, logistics, weather forecasts, and other information.

The goal of this thesis is to investigate the feasibility of replacing the recently developed graphical output display and the separate form-based input interface with one combined interface. The result allows the players to make their inputs directly through the graphics display. The new interface is be easy to use and learn, reducing distractions from the primary purpose of the exercise.

1.1 Background

The TWX was developed by the Air War College between 1976 and 1977. The exercise was meant to instill an understanding of the factors involved in airpower employment decisions. Emphasis was given to simulating the difficulties of coordination between the United States and Allied forces in a credible manner and of planning and providing logistics support. Also, the exercise was designed to give the participants a feeling for the problems involved in obtaining and processing vital information in order to make timely and sound decisions in a wartime environment.

The TWX is conducted over a four day period with multiple independent seminars being run simultaneously. Each seminar consists of two teams, Blue and Red, matching strategy in Central Europe. The Blue team is made up of eight to ten students, who represent NATO forces and the Red team is made up of faculty and AFWC personnel familiar with Warsaw Pact strategies. Also there is a White (or Control) team, consisting of the faculty game director and data processing personnel who oversee the conduct of the exercise.

A typical period of play consists of the players analyzing the current situation displayed by the exercise, developing their strategy in response to the situation, and entering their decisions into the program. The batch portion of the program which uses the players responses and information from the database (to determine the battle's outcome) is then run. The database is updated and a new output display is generated. The information in the database remains static during the day except for aircraft movement, aircraft rerole, logistics movement, and player responses which are stored for batch processing between exercise days (2).

From 1977 through 1987, the TWX was run on a Honeywell 6000 series mainframe computer. The input was done through slow teletypes which limited the interaction by game controllers and exercise participants. Players had to learn a complicated computer syntax, could not receive on-line help, and had no way to correct data input errors. The output was a voluminous stack of computer printouts that had to studied to gather information for the next period of play. In 1987, two Air Force Institute of Technology (AFIT) students, Michael Brooks (5) and Mark Kross (15), rehosted the exercise from Honeywell to a DEC MicroVAX using Zenith Z-158 microcomputers as remote terminals.

In addition to rehosting the exercise, Brooks and, later, Ken Wilcox (27), rewrote the game controller to allow multiple seminars of the game to be controlled concurrently and redesigned the database from application-specific files to a commercially available relational database management system (DBMS) called INGRES. The rewrite of the source code made the system easier to maintain and modify.

Kross developed a new input interface on the Z-158's using a fourth generation language provided by INGRES. This interface eliminated the old hardcopy terminal and was designed to be easy for the players. The new interface also provided input facilities with immediate validation, feedback, and reinput capabilities.

In 1988, another AFIT student, Darrell Quick (18), developed a map-based graphical display to replace the large, hard to read stack of computer printouts. This interface showed the current status of forces over the entire exercise theater. The players are allowed to 'zoom' in on bases of interest to reveal status of forces and logistics information. In addition the players can overlay weather forecasts on the map.

1.2 Problem

The success of the exercise is not determined by the battle's outcome but on the lessons learned. The exercise's purpose is not to teach the students how run the computer exercise, but to grasp the complexities and pressures involved in making command level decisions. The students are assumed to be inexperienced in operating computer applications and "a good user interface makes the program not only easy to learn but also easier and more efficient to operate" (16:13). To ensure that the goals of the exercise are met, the interface must incorporate human factors in its design. When successful, the interface promotes confidence, satisfaction, and efficiency.

The graphical output is capable of showing only the current status of the theater of operations. This is useful in identifying existing threats and constraints. The exercise player must then change to the input forms interface to move aircraft and logistics and plan missions. The switching between interfaces promotes confusion and entails more work for the students than necessary. A combination of the two interfaces allows the players to view the current situation, and make and input their decisions at the same time. This system is more natural and easier to use for the players.

The new interface could incorporate additional features such as:

1. Updating the screen to show the movement of aircraft and logistics and allow the players to change any of these decisions before final commitment of forces.

2. Highlighting areas of concern through graphical means. These areas of concern would include bases where logistics shortages or overages exist, critical levels are being reached, and those that were hit hard in battle.
3. Including base IDs in the display. Currently a base cannot be identified unless the player keys on the base and uses the 'Show Detail' function. These IDs should be visible at the appropriate level so they do not clutter the display and assist the players during the exercise.
4. Defaulting to the 'Show Detail' function when the user keys on a base. This should be restricted as to not interfere with base selections for the input and display functions.
5. Incorporating a password system. The graphical output display allows the player access to all the information displayed on the map. This is fine for the control team but the other teams should not have indiscriminate access to display and control of the opposing forces weapons and materials. Information available to the player on opposing forces such as location, type, and status is based upon the number and type of intelligence-gathering missions flown. Manipulation of aircraft and logistics is restricted to the player's own bases.

1.3 Assumptions

The development of the input/output interface and related modifications is based on the following assumptions:

1. AFWC was satisfied with the graphical output display providing the password system is implemented.
2. The new interface will be accepted if it produces results consistent with the current TWX system given the same input parameters and performs to the satisfaction of the AFWC.

1.4 Scope

This project's purpose was to combine the graphical output display and the form based input into one, easy to learn and use interface. The project includes a review of existing graphic software packages to determine which was most suitable for building this interface.

The TWX system is being rehosted at AFWC onto SUN 386i workstations. AFWC prefers the database be implemented with the commercial ORACLE DBMS. Due to the availability of graphical software this interface was written using the HOOPS Graphics Software. Transfer to the SUNs and the ORACLE DBMS requires only the replacement of the test files' processing statements with ORACLES Structure Query Language (SQL). These limitations should not restrict the design of the new interface or the final recommendations of this project. The selection of HOOPS as a graphics toolkit by Captain Quick was sound and was still valid for implementation on the workstations (see Chapter III). This project is limited to transferring the AAFCE input functions to the new interface.

1.5 Approach/Methodology

The enhancements to the TWX were limited due to the amount of time available for implementation. Each enhancement had to be prioritized and then developed, evaluated, and implemented separately. This is the iterative approach where successive versions of the system are built, each incorporating an additional modification from the previous version. This produced a product ready for implementation and there was always a stopping point readily available.

Within the framework of this iterative design each version was developed using the classical life cycle approach, which involved functional specification, preliminary and detailed design, implementation, unit testing, and integration testing. Full system installation was delayed until the ORACLE database was loaded onto the SUN workstations.

The modifications were prioritized and implemented as follows:

1. Password system - this enhancement is necessary to provide a usable graphical output display and a necessary component of the proposed graphical player input interface.
2. Combining the graphical output display and the formed based AAFCE input functions - this was the major purpose of this thesis effort.

3. Developing a graphical report of critical areas to augment the current form based reporting.
4. Add base identification on the graphical output display.

1.6 Sequence of Presentation

This thesis consists of six chapters. This chapter has presented an introduction to the TWX system and the proposed modifications. The next chapter reviews current literature related to the design of graphic interfaces. Chapter III describes and compares different graphic software systems available. The subsequent chapter describes the methodology used in implementing the project. Chapter V details the requirements gathering, followed by a chapter elaborating on the design issues. Chapter VII details the finished prototype. The last chapter presents conclusions and recommendations for further research.

II. Review of Literature

It can be confidently stated . . . that the user interface is becoming a key aspect in software design. This will be especially true as the number and diversification of inexperienced users continue to grow. (20:36)

2.1 Computer Interfaces

A computer program's success is dependent upon the user's ability to use it productively. In the early days of computer applications, the users were isolated from the computer systems. The early application interfaces consisted of a series of cryptic commands entered from a mainframe computer console. Because the system operators were machine conversant, the trend was for software designers to concentrate on the machine end of the software and not on the user end. Their areas of concern were the efficient use of two resources — computer time and memory space. Human factors (ergonomics) was not a concern for these software designers.

With the advent of cheaper hardware and faster machines, and the emergence of the personal computer (PC), the role of computers in today's society has expanded, and more and more people are sitting down at the computer terminals. Programs, supposedly well designed, are not succeeding and these failures can be attributed to poorly designed or neglected man-machine interfaces. Ill-formed interfaces are not only difficult to learn but are also inefficient, even in the hands of an experienced user. "Ease of use, not ease of implementation, is becoming the crucial design consideration"(7:218).

2.2 Human Factors and Interface Design

Human factors are essential to user acceptance and useful interactive software. The fact that they are not currently in widespread use is surprising because the demand for better user interfaces is recognized. Supply has not met demand, despite the supply of talent capable of developing user interfaces has been building. Literature on user interfaces design issues has increased dramatically, and the methodology of software design has reached a point where the human factors consideration can be incorporated early in the design. (8) Competition between software development firms and the users, themselves, will correct this deficiency.

The user interface should promote confidence, satisfaction, and efficiency. Saja presents the development of such an interface as a blending of the software designer's conceptual model of a computer system and the user's cognitive or mental model. The conceptual model guides the engineer's design of this interface which, in turn, presents an image of the system to the user. If the two models do not coincide the result will be user frustration and disappointment. To avoid this, the designer must constantly keep in touch with the user and help shape the user's cognitive model. When the user receives the final product there should not be any surprises. (20)

Yestingsmeier (28) describes the designing and implementing interactive software as consisting of three human factors activities:

Study Phase Activity This phase brings the user into the design process early. The user helps the design team develop an accurate problem definition. An additional benefit of bringing the user in early is there will be less resistance to change later on.

Design Phase Activity Several human factors questions must be considered in this phase, such questions as:

1. Are there memory aids and help screens?
2. Is an on-line tutorial available?
3. Are the screens simple and uncluttered ?
4. Is the system protected from the user?

When answering these and other questions, the user is constantly consulted.

Development/Operation Phase Activities This is the final phase where the interface is implemented, clear and readable documentation is provided, and the user is trained in the use of the interface.

The end product, the interface, should have no surprises for the user. The interface should be modular, consistent, and provide constant feedback. If human factors are incorporated early in the design phase and used throughout its development, then a good interface should result. A good interface (4) includes:

1. Consistency.
2. Support for a wide range of users, from computer novice to expert.
3. Support for error handling and recovery - - - the system should be forgiving.
4. Support of modular and modifiable applications.

2.3 Icons and Direct Interface Manipulation

Interfaces have traditionally supported a human-computer dialogue, where the user types in commands to express the actions he wishes to be performed. The computer answers by performing the requested action or indicating some error has occurred. An alternative to this type of interaction is known as direct manipulation where "the user has the illusion of directly acting upon the objects of interest without the intermediary of the system" (11:120).

Quick cites the work of Hutchins, Hollan, and Norman in noting the psychological benefits of an interface where the user can directly manipulate icons (graphic representations of an object or action) (12). The typical interface, where the user types in commands for the computer to manipulate data, leaves the user feeling distant and not entirely in control. A direct interface displays the data as icons and the user can directly manipulate them by using a pointer device. This restores the feeling of control to the user.

Hudson (11) cites Shneiderman (21), who stated that a direct manipulation interface should have:

1. Continuous representation of the objects of interest.
2. Physical actions or labeled button presses instead of complete syntax.
3. Rapid incremental reversible operations whose input on the object of interest is immediately visible.

Hudson (11) goes on to summarize a direct manipulation interface as a system that addresses:

Engagement The feeling of communicating directly with the objects of interest.

Articulatory Distance The degree to which the form of communication with the system reflects the application objects and tasks involved.

Semantic Distances The degree to which the semantic concepts used by the system are:

1. Compatible with those of the user.
2. Can be used to easily accomplish the user's goal.

Quick employed these ideas in his graphical output display in a number of ways, such as:

1. Allowing the user to select bases and units, to display additional information, by using a mouse to select the icon that represents it.
2. Presenting the user a function keyboard, where the user can select a function with the mouse by 'pressing the button on'.
3. Representing the theater of operations as a map, with which the user can 'zoom' in on areas of interest and 'pan' across the map display.

2.4 Menu Design and Implementation

Menus provide an interface solution that incorporates the cognitive feature of recognition. They fill three interface requirements: increased productivity, improved interface facilities, and ease of use.

Menus allow the user to view the available options and select the ones they wish to use. These options are selected through a keyboard by a single keystroke or another input device such as a mouse. This type of selection is thought to be more natural to the user and speeds their learning because they are using the more powerful human capability of recognition rather than recall.

Good menus answer three design issues: option selection techniques, display design, and functional organization of selections. option techniques are dependent on the interface hardware available. Screen designs should be readable, simple and uncluttered, and the screens named as an aid to the user.

Functional organization consists of three factors (10):

1. The grouping of functions on a screen.
2. The number of options listed on the screen.
3. The hierarchy of embedded screens.

The menus that group similar functions in a logical grouping are more helpful to users in recall.

The number of items on the screen can affect a user's ability to interface, Hodgson reports that studies have shown that four to eight items per screen is optimum (10). These options are better used if they are ordered in some manner either alphabetically or logically.

Embedded screens are paths the user follows to get to a desired option. The paths descend from general descriptions of commands to the specific commands and are extremely useful with the novice user. Accurately named screens and user familiarity with the hierarchy enhances their performance. More experienced users should be allowed to bypass upper levels to reach the specific menus.

Tolle and others have reviewed and made studies to determine if depth (more levels of screens and fewer options per screen) or breadth (less levels and more options per screen) is better for the user. Their findings are that less depth and increased breadth are more conducive to user interaction. This implies that the number of embedded menus should be minimized but not at the expense of readable screens (22, 25).

III. Graphics Software Toolkits

In the early days of interactive computing, users punched in commands on typewriter keyboards and the computer printed out the results. As time went by the proliferation of video display terminals allowed users to save paper, but everybody still communicated with computers using letters and numbers. It was not until the introduction of computer workstations that the technologies of computer graphics, bitmapped video display, keyboard entry, and pointer entry were integrated into a single package. (14:1)

This chapter reviews and evaluates graphics software toolkits. It first examines standards and proposed standards for a graphics-application interface and then compares the most suitable, for the new TWX graphic interface, to the Hierarchical Object Oriented Picture System (HOOPS). HOOPS was used to build the current graphical output display and the prototype graphical player interface. Finally the X-Window System graphics capabilities, presently employed at AFWC for their graphic displays, are reviewed for applicability to the new TWX interface.

3.1 Background

Computer graphics has rapidly grown into a major branch of the computer industry. Applications have included cartography, computer-aided design, flight simulators, and wargames. The great advances in computer hardware, including the arrival of workstations, has led to a corresponding increase in their graphics capabilities. The users' demand for these graphics and the different graphic interfaces, inherent to each different vendor's machines, have increased the challenge to the application developers. Graphics software toolkits have emerged to simplify the developer's task. The development of these toolkits are analogous to the development of higher-level programming languages to replace machine-level code.

"A good graphics package simplifies the programmer's task and makes it possible to write portable programs that can be run on different computers and with different displays"(16:12). A 'good' graphics package has been difficult to develop due to the rapidly changing state of computer graphics hardware. To solve many of the problems, graphics experts recognized the need to develop a set of industry standards similar to the standards recognized for higher-level programming languages.

3.2 Graphics Standards

The driving force behind the development of graphics standards was the need to provide graphics packages that were portable from one computer system to another.(26:1-6). The development has been difficult because these standards must cover: input methods, graphics construction, graphics storage, and output displays. This has led to several proposed standards, all of which construct, store, or transmit pictures, defined by the basic graphic primitives such as points, lines, and rectangles and bit patterns.

The seven major proposed standards can be broken down into three functional categories (26):

Graphics-Application Interface. This category addresses the construction of graphics by a computer system. It includes CORE, the Graphics Kernal System (GKS), and the Programmer's Hierarchical Interactive Graphics Standard (PHIGS).

Storage and Transmittal of Graphics Data. This category includes:

1. Initial Graphics Exchange Information which is concerned with the storage and transmission of pictures, and
2. Computer Graphics Metafile which presents a general purpose approach to the storage of a picture, regardless of how it was created or how it will be displayed.

Graphics-Device Software Interface. This category includes:

1. Computer-Graphics Interface whose goal is to create a universal interface between the upper levels of a graphics system and its device drivers, independent of the hardware.
2. North American Presentation-Level Protocol Syntax which defines the data transmission interface for hardware with very specific data-processing and data-storage capabilities.

Currently the graphical player interface is only concerned with the graphics-application category.

3.3 Core, GKS, and PHIGS

Core, GKS, and PHIGS are all graphics toolkits which support the two-dimensional graphics necessary for the TWX graphical player interface. They also provide one or more virtual coordinate systems and viewing operations so the graphics programmer does not have to be concerned with the specific graphics hardware.

3.3.1 Core. "As the first proposed graphics software standard, Core has established terminology and concepts that are reflected throughout the graphics industry" (26:2-1). Core established the concept of two-level graphic structures, called segments, which can include dozens of individual primitives. These segments can be transformed or displayed as single entities.

Core's definition of interactive inputs set a pattern for all subsequent standards (26). Core defined six input classes (a single input device, like a keyboard, can represent several input classes) and three input modes.

The six input classes are:

Locator. This input is used to identify a position in Normalized Device Coordinate (NDC) space.

Evaluator. This input supplies a value type measurement.

Stroke. Stroke devices generates a series of positions in NDC space.

Button. Also called 'choice', this class allows the operator to choose among a set of options.

Keyboard. This device is used to input of alphanumeric characters.

Pick. Pick devices allow the operator to identify a particular primitive or segment.

The three input modes are:

Request. This mode requires a timed interaction between the application and the user.

Sample. The application program can sample the input device at any time without waiting for the operator.

Event. This reverses the process the operator can input information at any time and the information is placed in a queue to be processed by the application at its convenience.

The new interface uses every class of input, but only requires one input mode - the request. This is the only mode necessary because the graphical player interface is primarily a decision making aid and is passive unless reacting to a user input. In other words, the user makes a decision on what he would like to do, inputs that decision using the methods provided by the interface, and the application responds. Once that response is made, the application waits for more input. There is not any need for the interface to actively look for inputs or to place user inputs into a queue to be processed at a later time.

A problem with Core is the lack of any standard bindings to an upper level language. Application portability is limited to installations with similar or identical Core-based software packages (26). GKS and PHIGS provide standardized bindings to FORTRAN, Ada, Pascal, and C. The current output and the new prototype are written in C.

3.3.2 GKS. GKS is related to the Core system and is a logical progression from Core. Like Core, GKS relies on the same basic primitives and the segment. Both model objects in world-coordinate space and convert the view to NDC space. Both standards support the six input classes and three input modes. GKS and Core do have some basic theoretical differences in addition to the language bindings, "all of which address perceived Core weaknesses" (26:3-1).

Higher levels of GKS add two important new primitives. One is a cell-array primitive that permits the coding of individual pixels or groups of pixels, and the other is a generalized drawing primitive (GDP) that allows the programmer to take advantage of known hardware, firmware, or software capabilities of an output device. Both of these primitives hold no interest to the new TWX interface at this time.

Another difference is the handling of logical workstations. Core presumes that the output will be displayed on a single, known output device or devices with very analogous characteristics. GKS can connect and drive multiple independent graphical workstations. "A workstation is, e.g., a plotter or a display with a keyboard or tablet connected to it. The workstation concept is one of the original contributions of GKS to the methodology of graphics system design" (6:12)

3.3.3 PHIGS. "PHIGS utilizes and builds on many GKS concepts, including the basic primitives and their attribute types, the logical input device model, and the workstation concept" (1:13). But PHIGS is an extension from GKS (just as GKS evolved from Core). "PHIGS, by comparison [to CORE and GKS], is truly a programmer's toolbox. In fact, its functions are called elements, and in many ways are equivalent to a higher-level programming language" (26:4-2).

Core and GKS support primitives and segments, while PHIGS supports primitives structures, and substructures. The difference is that when attributes and/or transformations are applied to a segment, they are applied to the whole segment. Also, once a segment is closed primitives cannot be added to it without its deletion and recreation. Attributes are bound dynamically to a PHIGS structure. A structure, or its associated substructures and elements (primitives) can have transformations applied or attributes changed at any time. A structure primitive may be a graphics primitive, an attribute or view selection, a transformation matrix, or an execute structure element (invocation of another structure). Attributes applied to the parent are inherited by its children, while attributes applied to the children do not effect the parent. Structures are arranged in hierarchical structure networks and are editable: elements may be inserted and deleted. PHIGS minimizes the need of redefining data in order to modify it.

The PHIGS structure is more suitable for the graphical player interface than the Core and GKS segment. Rather than think of the map display as a collection of separate segments (such as boundaries, coastlines, red bases, red units, blue bases, and blue units) it is preferable to envision the display as a hierarchy, with the window, in which the map is displayed, as the parent and the other components as children. Actions, like enlarging a section of the map, can be applied to the parent window and, subsequently, to all its children with one command. Individual components such as the red bases can have their visibility turned on and off, without affecting the rest of the display. The map can be manipulated as a whole or down to its individual bases with ease.

3.4 HOOPS

HOOPS is the graphics toolkit used for the TWX graphical output display and the new graphical player interface prototype. It was used by Quick for the following reasons (18):

1. HOOPS supports the workstation environment and PC's and the applications are portable to any supported configuration with no changes.
2. HOOPS provides standard bindings to a variety of higher-level languages, including Microsoft C. This makes HOOPS compatible with the TWX application, because TWX uses the Ingres Embedded Structure Query Language for data retrieval and Ingres SQL supports several language bindings including support C.
3. HOOPS is available, under site license, so it is available without any additional expenditures. AFWC, however, would have to purchase HOOPS to maintain the software.
4. HOOPS offered features not available in other toolkits including a Hewlett-Packard Graphics Language plotter driver, a Postscript laser printer driver, mouse drivers for each supported terminal device, and two high resolution graphics drivers for the PC version.

Quick stated that a disadvantage of HOOPS was that it did not conform to any major graphics standard (18:29), but John Aronson, of Ithaca Software, the developers of HOOPS, stated they hoped to become a de facto standard in time (3).

HOOPS can be considered a step above PHIGS in the evolutionary scale. It is an object oriented toolkit not a procedural one. HOOPS approaches graphics applications by first assuming that "nearly all graphics applications are really database problems" (13:1).

HOOPS is an extension of the previously mentioned toolkits, in that it supports the same basic primitives (geometry), input methodology, and attributes. The HOOPS' objects are defined and stored in an object library that is organized as a tree-shaped hierarchy, resembling a file directory. The system provides the application program with tools for modifying, querying, searching, and displaying this graphical database.

The unit of organization within this database is the segment. A segment is a collection of related elements, such as geometry, attributes, and other segments, which are grouped together for easy handling. The HOOPS segment is more akin to the PHIGS structure rather than the GKS and Core segment, in that it can contain primitives or other segments (which results in the tree-like structure) and any part of it can be modified at anytime within the application. The segment's attributes have the feature of inheritance like PHIGS and is controlled in the same way.

The most attractive feature of HOOPS is the straightforward way the language presents itself. A routine name consists of an active verb, to indicate what is being done, and the object of that action. An example of this is Set_Visibility("blue_bases", "off") which would render the blue bases invisible on the screen. This lends itself to the object oriented programming style that the language Ada presents and is advocated by the Department of Defense.

HOOPS provides many functions that lend themselves to cartographic displays, not included in other graphic toolkits. Functions such as zooming and panning would require complicated routines developed by the application programmers.

HOOPS is recommended for the final implementation of the combined TWX interface because:

1. HOOPS is portable. The interface can be developed on a personal computer or on VAX/VMS system and then be ported to the SUN workstations and the X Window system that AFWC now employs and advocates, without modification to the HOOPS code.
2. HOOPS allows the TWX map to be defined as a collection of parts, where attributes and transformations can be applied to a single part or to the whole collection, dy-

namically.

3. HOOPS is a graphics toolkit that provides higher level routines that eases the work of the developers of interactive cartography graphics.

4. HOOPS, as a language, is simple and supports object-oriented design.

3.5 X Window System

"The purpose of the X Window System is to provide a network-transparent and vendor-independent operating environment for workstation software"(14:4). Network transparency means that computer systems ranging from stand-alone workstations to time-shared mainframes, using many different workstations as if they were terminals, can be used to display the same application software. The network transparency of X doesn't favor one style over the other, but "synthesizes them all into a flexible networked computing environment suitable for use anywhere"(14:5).

X applications are portable, they deal with X so they don't need to see the particular workstations display hardware. As long as the application is able to establish a connection to the workstation then it can use the full capabilities of the base window system.

X offers the following graphics capabilities:

1. Organizes display screens into a hierarchy of overlapping windows. Each application can use as many windows that it needs, resizing, moving, and stacking them on top of one another as needed.

2. Drawing capabilities that include the basic graphic primitives. These drawing operations, though, are immediate, in that there is no memory of what was drawn, only that certain pixels are lit. The drawing operations are bitmap-oriented, that is the applications specify all operations in terms of integer pixel addresses within windows.

3. Lets applications draw high quality text.

4. Useful to a wide variety of monochrome and color workstations.

5. Useful for displaying, manipulating, and capturing images.

The central purpose of the X Window system is to allow many different applications to be active on a workstation. The primary tool provided by X Window system is the window, a rectangular area on the screen of a workstation. X makes windows, plentiful and inexpensive, so the application programmer can create as many windows as he needed. The programmer can use the screen like a desktop with pieces of paper, each piece representing an application. Usually (but not always) each application uses a separate window or window sub-hierarchy to display its output. Each application operates separately and doesn't need to take heed of other applications; X takes care of relaying user input to the appropriate application. X's goal, with displaying these applications, is to provide a level of interaction with the user, that is smooth and natural. X provides a high level toolkit subroutine library that the application developers can use to ease and standardize these interactive responses.

The X windows are stored in a hierarchy where the root window (the screen) is the only window on the screen without a 'parent window'. Each new window is created in the unmapped state and a window and its ancestors must be mapped before it can be seen. X provides library routines to make the creation, mapping, soliciting events, unmapping, destruction, and the changing of the borders for these windows simple.

It is this windowing system and its event handling that makes X an attractive system to employ. The system does insulate the user from the hardware, supplies the basic primitives needed, and provides the capability of specifying how these primitives are to be displayed.

To specify the attributes of a primitive, such as color, the X Window System provides a resource called the graphics context (GC). The developer can create as many GCs as he requires and one GC can be used to define the attributes of many segments. The GCs are not tied to a primitive but to a user defined window.

The basic tools are included in X to develop the user interface but it does not have the high level routines or ease of style that the HOOPS toolkit does. After the graphic is drawn with X, the system retains no memory of the graphic, only that certain pixels are lit on the screen. X graphics are procedural and not object oriented, there are no structures or segments to store in the program and display when needed. A good example to show the difference between X graphics and HOOPS would be drawing the blue bases and then changing their color for highlighting. In X there would be a procedure Draw_Blue_Bases with a GC specified for attributes (this would have to be set earlier). HOOPS users would open a segment called 'blue_bases' and then would define the attributes. So far there is very little difference between the two, but when highlighting the bases, the X user would have invoke the Draw_Blue_Bases routine again with a new GC or the original GC changed. The HOOPS programmer could use the 'quick' command HC_QSet_Color('blue_bases','highlighting color'). The work done by the computer is probably comparable between the two operations, but there is a difference in the work done by the system developer with X.

Still AFWC has successfully adapted the graphics to their wargame StratWar. This is an interactive game but the interaction is confined to buttons and menus, which are easily developed with the X Window system. There is no direct manipulation with the map display. Interaction with the actual map display is limited to choosing an area to zoom in on and recalling a procedure to redraw that area at a larger scale. This zooming function could have been implemented easily with the HOOPS toolkit.

The X Window system could be adapted to make the TWX interface because the X system is an excellent tool to use to build buttons and menus for the user interaction. But there would be an increased burden on the applications programmer that would not be there with HOOPS when developing the procedures to display the components for the TWX map and simulate user interaction with them.

HOOPS simplifies the graphics developers task and can be used to augment the X capabilities, because the systems are compatible. Combining the X Window System and the HOOPS toolkit presents some attractive future modifications. One future enhancement would be the creation of a Red player expert system. A separate X window containing the Red input functions would be started from the graphical display and parameters could be passed between the two windows. The input window would present a less rigid environment to the experienced Red player. A key feature of the X Window Systems is that functions running in separate windows can pass information back and forth and in effect can 'talk' to each other. This potential flexibility leads to the recommendation that HOOPS be used under the X Window System.

IV. Design Methodology

Before work can proceed on any project an outline guiding the development of this work must be produced. Kross and Quick reviewed several software engineering paradigms including the three that showed promise for the graphical player interface - - - the classic life cycle, prototyping and iteration.

4.1 Classic Life Cycle.

The classic life cycle paradigm calls for a systematic approach to software development. The steps in this approach include (see Figure 1) (24):

Requirements analysis and definition The system's services, constraints, and goals are established and defined.

System and software design Using the requirements definition as a base, a design is drawn up of the system that can be readily transformed into a computer program.

Implementation and unit testing The software design is transformed into software units. These units are tested to verify that they meet the specifications.

System testing Units are integrated and tested as one complete unit. After the testing is completed and successful, the software is delivered to the user.

Operation and maintenance The software is installed and put into use. Maintenance occurs throughout the systems lifecycle.

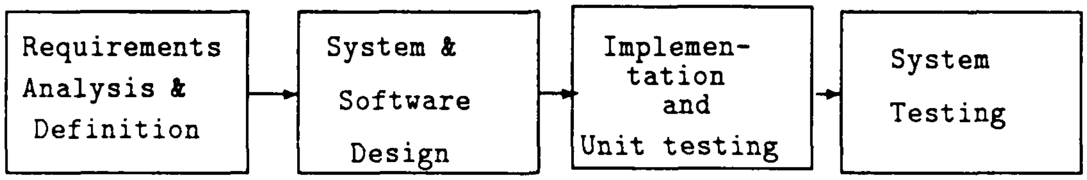

Requirements Analysis & Definition
System & Software Design
Implemen-tation and Unit testing
System Testing

Figure 1. The Classic Life Cycle

In actual practice these phases are rarely distinct; they can overlap and feed off each other. A major problem with this approach is that the system development rarely follows the sequential flow. Iteration always occurs and creates problems because it is difficult to determine the projects progress. Another major problem is that it is often difficult for the user to state all the requirements up front.

4.2 Prototyping.

The prototyping paradigm can be used when the user does not have a complete set of requirements. An initial system, which may lack any support processing, is developed by the designers to demonstrate to the user that the project is feasible. This initial system can take three forms (17):

1. A paper prototype that depicts human-machine interface in a form the user can understand.
2. A working prototype that implements some subset of the system.
3. An existing program that emulates part of all of the functions desired but needs to be improved upon for this development effort.

This paradigm consists of (see Figure 2) (17):

1. Gathering requirements,
2. Working a quick design.
3. Building a prototype.
4. Evaluating and refining requirements.
5. Engineering the product.

This approach, like all others, initially starts out with requirements gathering. After this initial phase, a 'quick design' follows which focuses on those aspects of the system that will be visible to the user. This quick design leads to the development of a prototype that is evaluated and refined; a process of iteration which continues until the system fits user requirements.

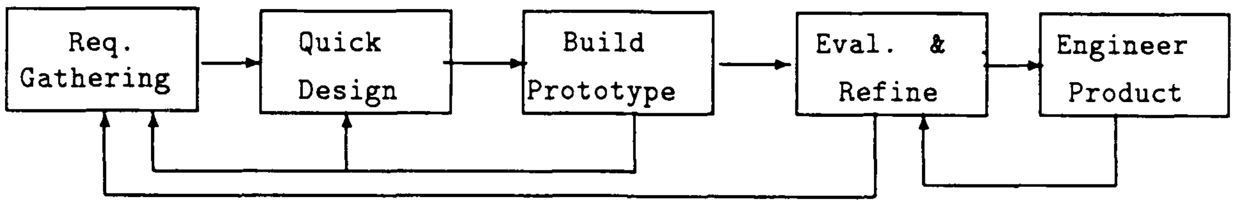


Figure 2. Prototyping

The benefits of prototyping include (24):

1. Misunderstanding between software developers and users may be identified when the initial prototypes are presented.
2. Missing requirements may be detected.
3. Confusing user services may be identified and refined.
4. Incomplete or inconsistent requirements may be identified as the prototype is developed.
5. A working, though limited system is available quickly, to demonstrate the feasibility of the application.

Prototyping does have some problems such as important features are usually missing from the interface support software and are not evaluated. Also reliability, robustness, and safety cannot be adequately expressed. Still the TWX graphical player interface was a prime candidate for prototyping, because it depends on visual displays and heavy interaction with the user (17). There was one drawback to prototyping the player interface and that was the large amount of complex code that made up the output display interface and the form-based input interface. Fortunately this code could be partitioned into manageable pieces to develop prototypes for portions of the system.

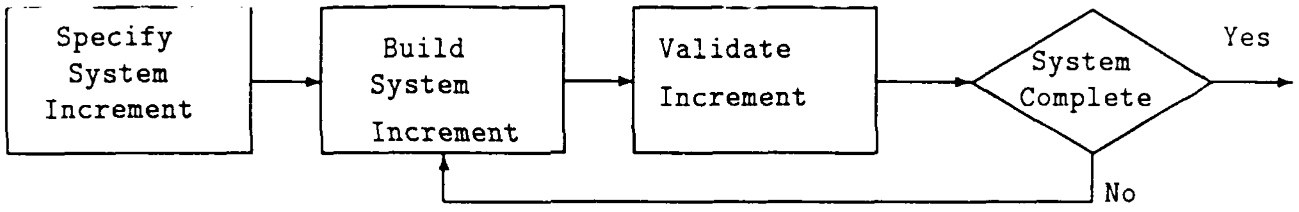


Figure 3. Evolutionary Development

4.3 *Iterative Design.*

The key point to this paradigm is that the system is broken down into small manageable pieces, which are prioritized, and implemented one at a time. This was attractive for this effort because of the time constraints involved. A workable system is developed at each stage and is ready for implementation. A version of this discipline called evolutionary development (24) is depicted in Figure 3.

Iterative design and evolutionary development consists of (15):

1. Identifying an important sub-problem.
2. Developing a small but usable system to assist the decision maker.
3. Refining, expanding, and modifying the system in cycles.
4. Evaluating the system constantly.

Another feature of this design is storyboarding to aid in the requirements gathering. Storyboards are paper depictions of the workings of the system and is similar to the paper prototype.

4.4 *The Methodology Decision.*

In truth a combination of the paradigms was used to develop the graphical player interface (see Figure 4). The first phase of the design was the requirements gathering to build the system specifications. Then each objective of the interface was prioritized and a quick design was drawn up on paper. The objectives were then developed as separate prototypes, supported by a skeleton of the output display interface. These prototypes included all the support processing necessary to implement the particular objective except the ORACLE SQL statements. In place of the SQL statements were flat file processing statements which accessed test data files.

After each prototype was evaluated, tested, and refined they were integrated into a single prototype on the SUN workstations. This prototype was supported by a more complete output interface but still accessing the test data file. The primary purpose of this prototype was to accomplish integration testing. Final integration and implementation testing will take place when the ORACLE DBMS is in place on the SUN workstations. At that time the flat file processing statements will be replaced by the ORACLE SQL.

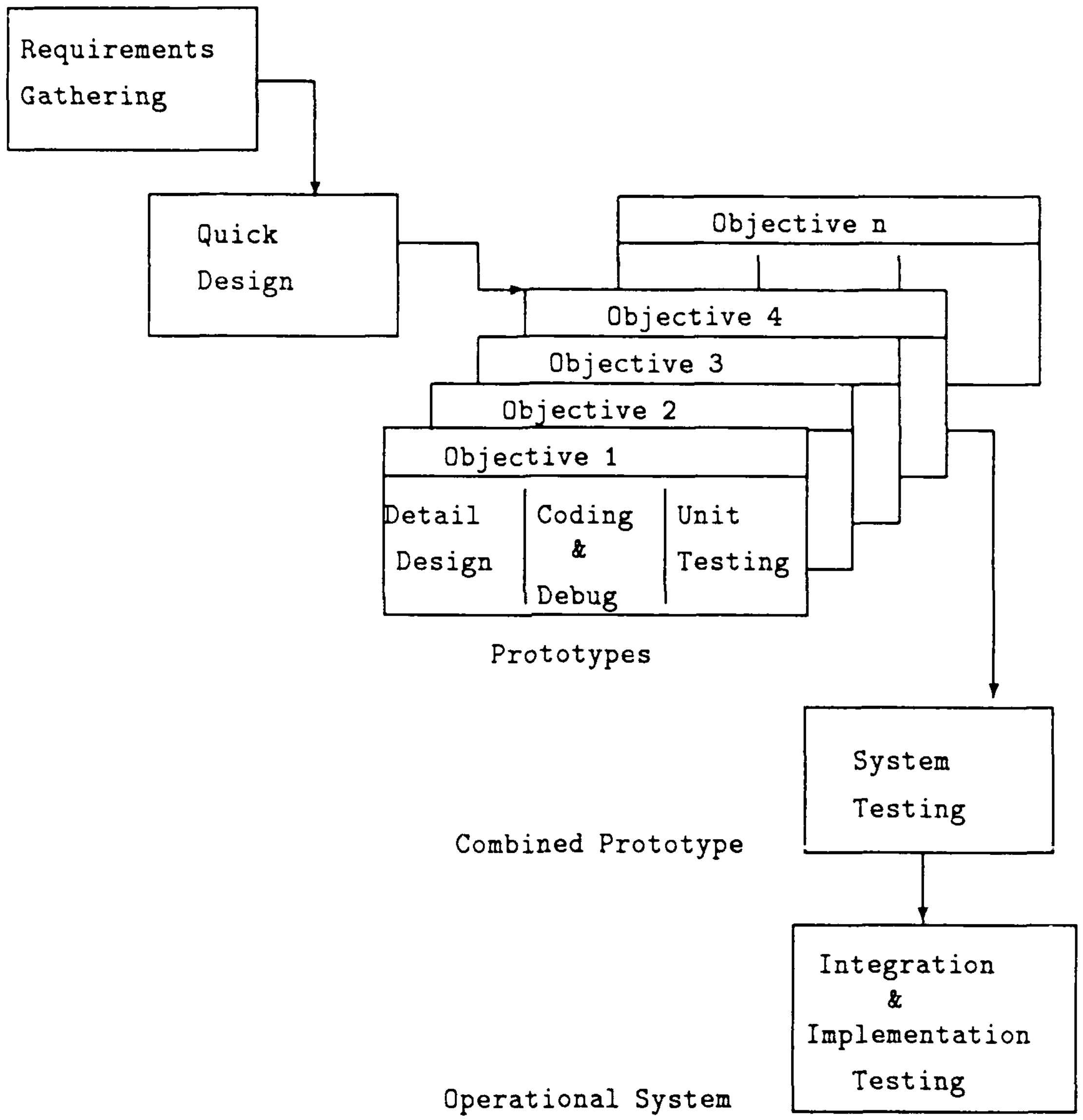


Figure 4. The Combined Methodology

V. Requirements Analysis

Each software engineering paradigm, or model, described in the previous chapter started with an initial phase of requirements gathering. The object of defining a new system's requirements is to assemble an overall picture of the inputs, outputs, operations, and resources required by the new system. Another goal of this phase is to define the evaluation criteria which will be used to evaluate the new systems performance. (24)

While the design model provides the 'how' for system development, the requirements gives the development the 'what'. This chapter describes the requirements analysis involved with the graphical player interface.

5.1 Given Requirements

The original thesis proposal stated the goal and enumerated the objectives to be accomplished toward achieving that goal. The goal of this thesis was to investigate combining the two current interfaces so that the player can make inputs directly via the graphical interface. The steps toward achieving this goal included:

1. Perform a literature review of graphical user interfaces. Determine the best methodology for this problem.
2. Design and implement the necessary extensions to the existing graphical interface.
3. Document the system with a user's manual and a system integrator's handbook.

Mark Roth (19), this thesis' advisor, also expanded on the requirements, by giving a set of additional modifications:

1. Including base and unit IDs in the display.
2. Defaulting to the 'Show_Detail' function when it would not interfere with another function's operation.
3. Incorporating a password system to restrict display and input functions.
4. Filtering the opposing team's data based on intelligence levels.

Gathering these general requirements was relatively straightforward. More specific requirements had to be gathered through research. A constraint to the requirements gathering process was the distance of the user from the developer. The design of the new TWX interface could not incorporate the user in the early stages. To overcome this, incorporating ergonomics in the design was given priority throughout the interface's development.

5.2 Human Factors

Through the literature review in Chapter II, a set of requirements governing the display and user interaction were assembled. A fundamental requirement of the interface was the preservation of system integrity. The user had to be protected from the system and the system had to be protected from the user. The interface had to promote confidence, satisfaction, and efficiency in the user. The displays had to be familiar and nonthreatening; providing encouragement for the player to use.

The input mechanisms had to allow the user to implement his decisions accurately and easily. The interface had to protect the user and system from input mistakes, and the user had to be able to recover from those mistakes. The user must have a clearly defined control flow including easy entry, movement through the interface, and exit.

5.3 AAFCE Input Requirements

This project was not purely developmental because it entailed modification of an existing system — the output display interface, and the rewrite of another — the form-based input interface. In order to develop a full set of requirements all the documentation and code on the two existing systems had to be gathered and reviewed.

After reviewing the documentation, the next step was to analyze Kross' and Quick's interfaces by generating structure charts of the software. These structure charts were used to determine the overall structure of the programs and identify important subprocesses. After these key processes were identified, they were examined in detail to distinguish key data elements and algorithms.

In rewriting the AAFCE functions of Kross' interface, the functionality and processing requirements had to remain the same. There were no requirements for the actual play of the game or for the decision-making algorithms to change.

The structure chart of Kross' interface (see Figure 5) helped identify the AAFCE functions and other introductory functions that had to be rewritten. The AAFCE functions fell in one of two categories — reports or inputs.

5.3.1 Report Functions. The primary consideration for implementing the report functions was that the requested information had to be presented in a readable format. The player, in addition to being able to request the type of information he wants, had to be given control over the amount of information (scope) displayed and the output format (printer or screen) of the display. These functions included:

1. Beddown Display.
2. Sortie Summary.
3. Logistics Capacity.
4. Logistics Inventory.
5. Predirect Rates.
6. Predirect Analysis.

5.3.2 Input Functions. These functions allow the user to manipulate data associated with their team's airbases. The information the user needs to make these input decisions had to be presented in a readable format. The input mechanisms had to be clearly defined and flexible enough to enable the user to implement his decisions. Every database change, the user makes, had to be given a gross error check before the changes were committed to the database. This error check was restricted to ensuring the data was the right data type and the value fell within an acceptable range. The system will not verify that the player made the 'right' move, doing this would defeat the purpose of the TWX exercise.

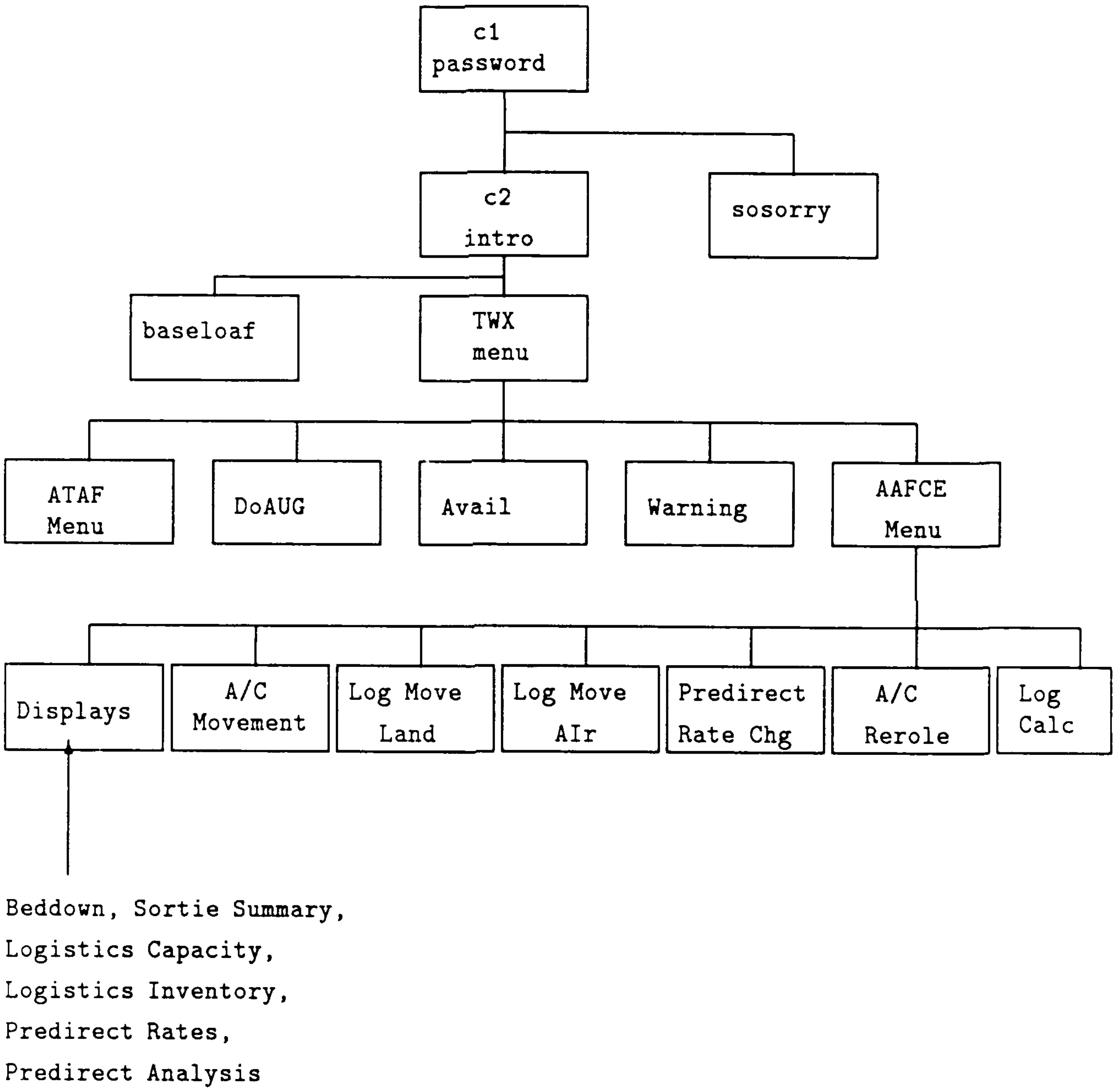


Figure 5. Structure Chart of the Form-Based Input Interface

The user had to have a sense of control over the interface and had to be able to change any decision before the actual commitment of database changes. The input functions included:

1. Aircraft Movement.
2. Logistic's Movement by Air.
3. Logistic's Movement by Land.
4. Predirect Rates Change.
5. Aircraft Rerole.

5.3.3 Introductory Functions. These functions were encountered in Kross' interface before the player was able to access the AAFCE input functions. The first was a password system to restrict player access and to determine which database tables to retrieve. Another function was 'Sosorry' which locked the player out of the input functions if the database was locked for batch processing. The last function was for calculating base loading factors and was not implemented in the prototype, but must be in place when the system is operational.

5.4 Output Display Interface

The output display served as a constraint to the design of the graphical player interface. The new interface had to be incorporated into the output display without any degradation to the existing system or the new system. To accomplish this the new interface had to 'fit' into the output interface's overall structure. The structure chart (see Figure 6, 7, 8) was essential to achieve this fit.

In keeping with Quick's design, the new graphics structures were all built and stored early in the program. This built a database of graphical objects for later use. The new interactive functions were all added to one subroutine, 'Manipulation', which was a loop acting on user input until explicitly exited. A hierarchy chart of the graphical objects (see Figure 9) in Quick's system had to be constructed to identify those objects that had to be modified and to distinguish where the new graphic structures should be added.

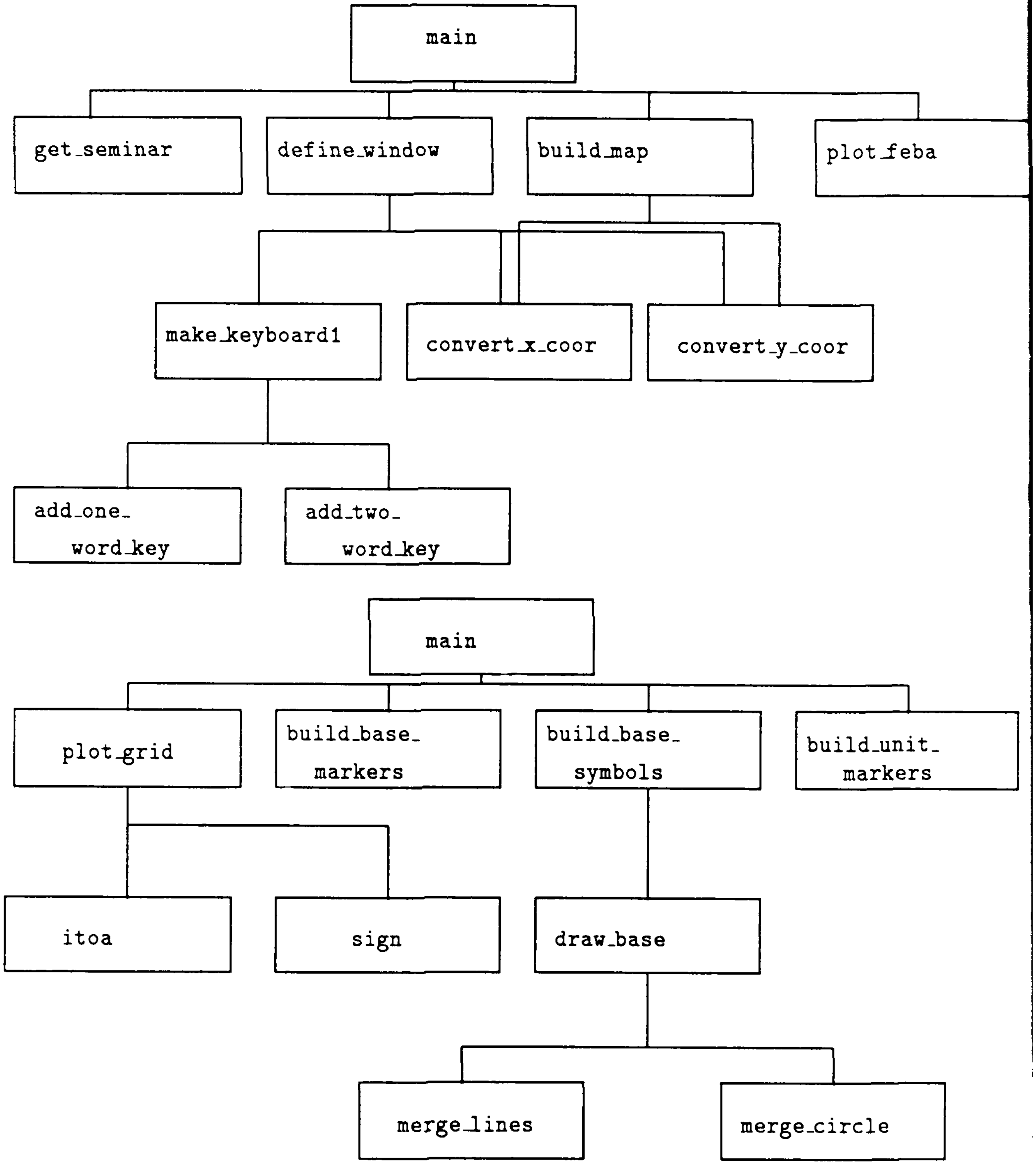
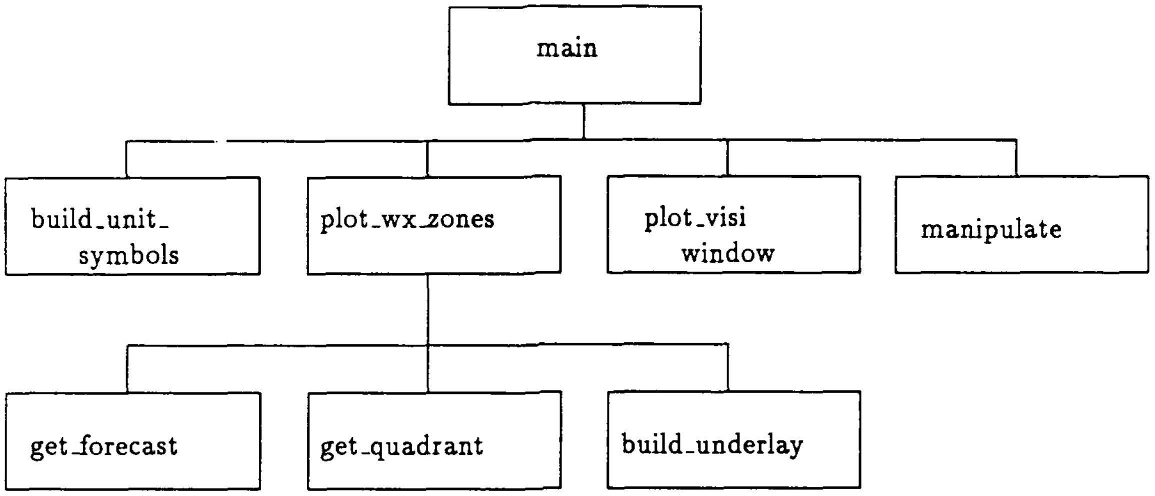


Figure 6. Structure Chart of the Output Display Interface - Part1



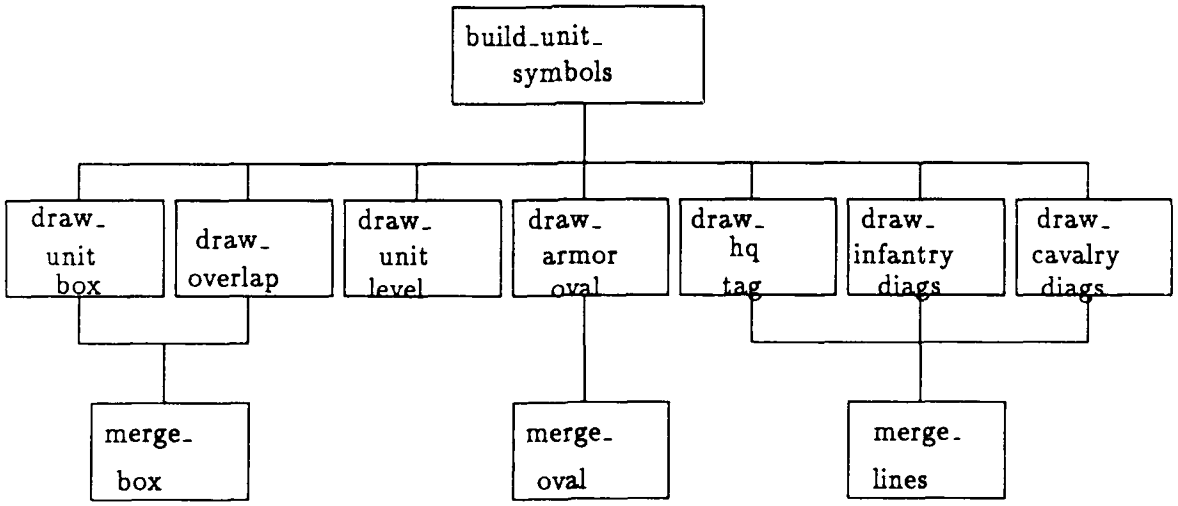


Figure 7. Structure Chart of the Output Display Interface - Part2

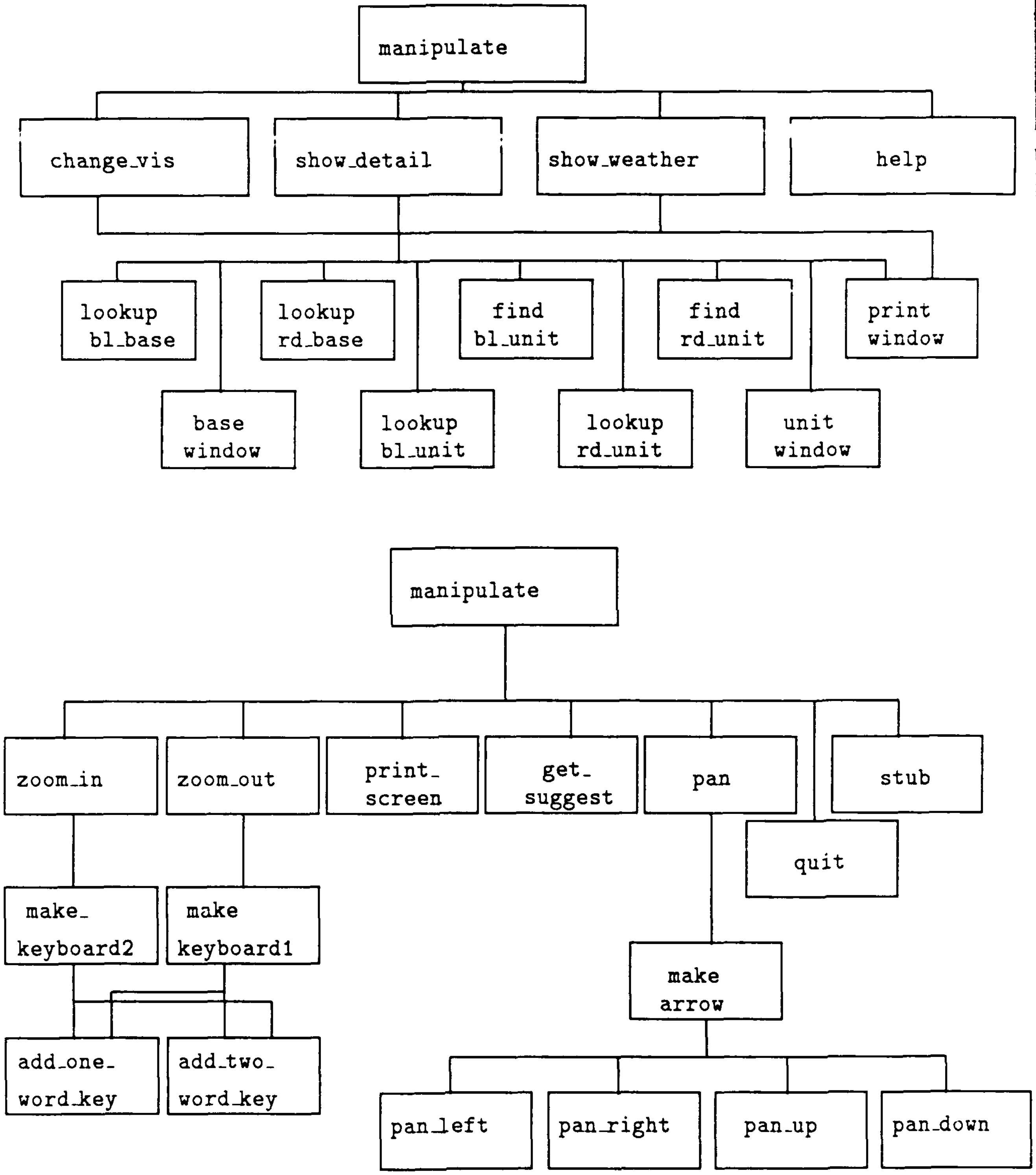


Figure 8. Structure Chart of the Output Display Interface - Part 3

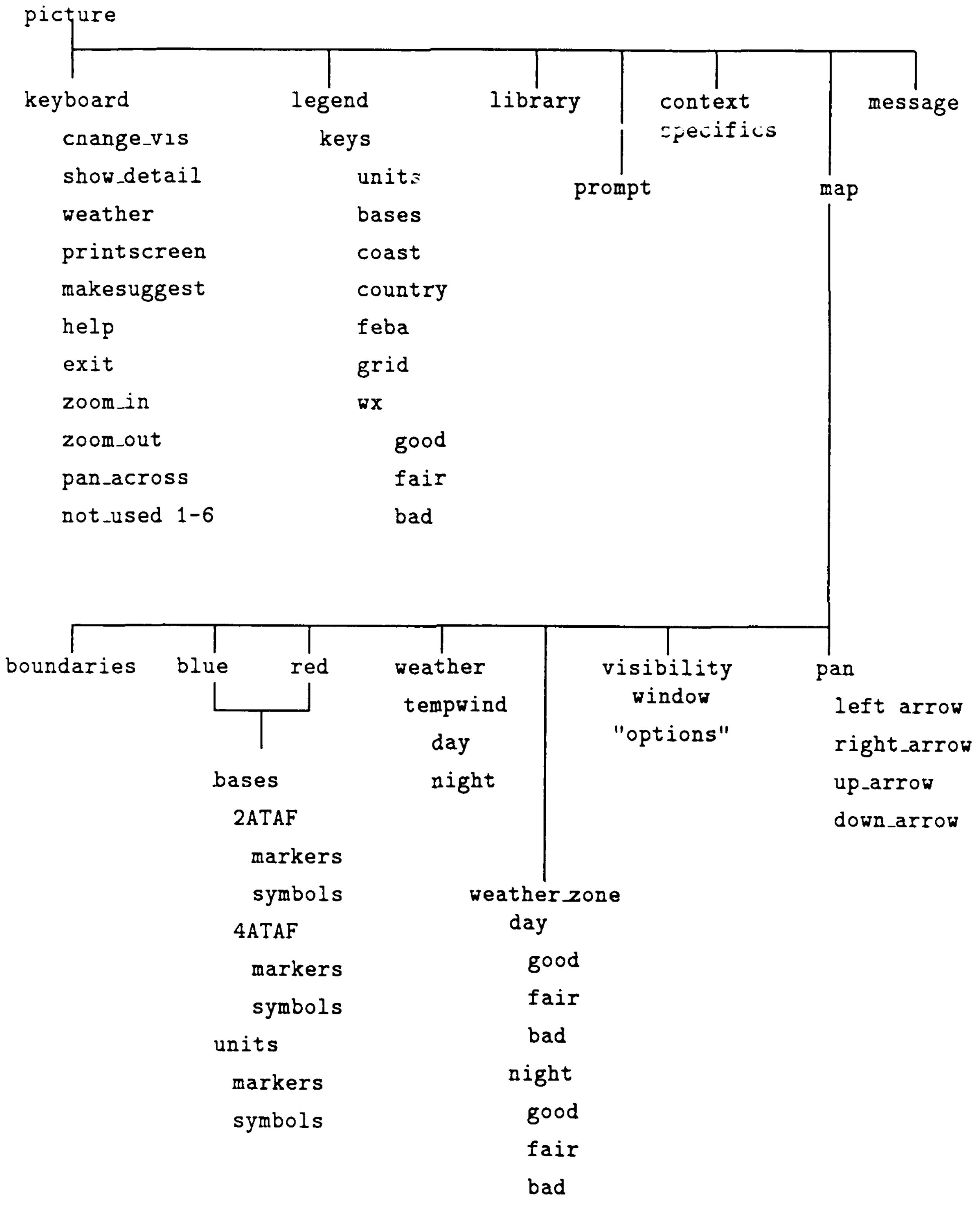


Figure 9. HOOPS Object Hierarchy

VI. System Design

After the requirements were determined, paper designs of the screen layouts were drawn and evaluated. These served as the starting points for the prototypes. Even though the prototypes did not have the full processing capabilities, they were useful in evaluating important aspects of the interface such as ergonomics. This chapter identifies several design issues in developing these prototypes and the decisions made.

6.1 Conception

Quick provided a conceptual model for his output display that was easy to understand and supplied logical extensions for the player interface. He envisioned and developed his model as a series of display levels, starting with the map and control area on the bottom and ending with the pop-up window level on top (see Figure 10) (18). The new interface is incorporated into the output display by utilizing existing devices at the lower levels and adding new mechanisms, as required, to the top level. Invocation of the new interface is done with the function "buttons" that Quick used for user control in his interface. The message area, in addition to feedback, is also used for user prompts and string inputs.

New extensions to the display include pop-up menus, display and input windows, and additional buttons. The menus allow the user to control the graphical player interface. The display and input windows are "work areas" where the system could displays data of interest and the user could make input decisions. The buttons would be additional mechanisms for control. In addition to the mouse, that is already used by the output display, the keyboard is used as an input tool.

Quick's groundwork left no question that the new interface was possible to develop with existing software tools and hardware. The challenge was developing a natural and easy to use interface, one that leaves the user in control of the system.

6.2 Graphical Display

The primary advantage of a graphical computer interface is the system can be modelled to better fit the users' backgrounds and perceptions. By presenting familiar control

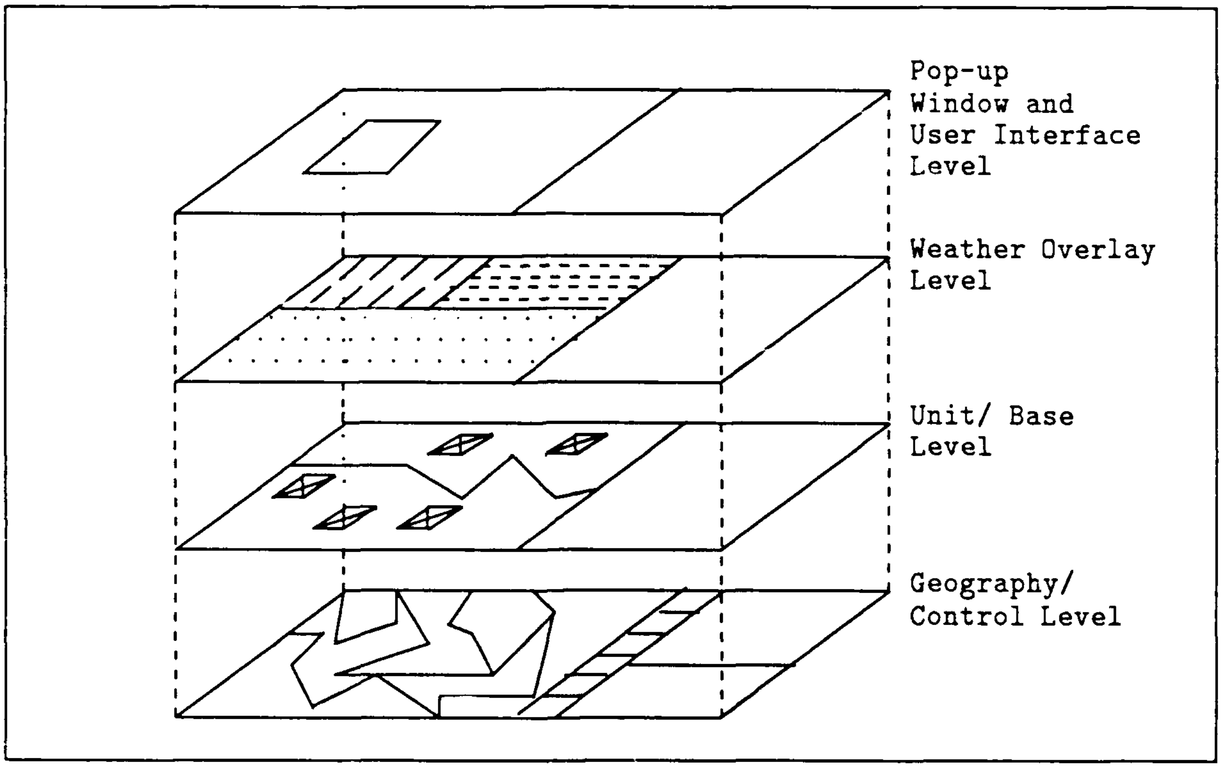


Figure 10. Conceptual Model of the Display and Interface

devices and displays, the interface effectively hides the computer aspect of the exerc The interface allows the exercise to better achieve its primary purpose of giving the players experience in making high-level wartime decisions.

6.2.1 Menus. Control flow is essential to any user interface. Control, in the graphical player interface, includs how the player enters the AAFCE input function, moves through the options provided, and eventually exits. How to implement player entry was obvious. Quick's function "keyboard" where all the output display functions were invoked, had four buttons that were 'not in use' (there were no functions associated with them). To avoid confusion with the hardware keyboard, this graphics device will be referred to as the display keyboard throughout the rest of this thesis. These unused buttons were ideal for the AAFCE input function and the three functions to be implemented later: 2ATAF, Land, and 4ATAF.

Upon invocation of the AAFCE function there was temptation to then replace the display keyboard with the next level of options. This approach was rejected because there were functions on the original keyboard, such as 'Help' and 'Print', that should be active throughout the exercise. As the player proceeds through the exercise there are, however, not enough buttons on the display keyboard to include all the functions needed. Also a continually changing keyboard confuses the user. Providing a static home for the 'Help' and 'Exit' functions gives the user a sense of stability and a place to turn when having difficulties.

Another possible solution was to include the options on an additional display keyboard. This had possibilities for the first level of options of the AAFCE input function, but there was also a second level for the report functions. The sortie summary report option presented still a third level of options, so a total of four keyboards are needed. This takes into account only the AAFCE function, the other three input functions may have more option levels when implemented. Displaying this many display keyboards may cause the user to lose track as to where he is in an input function and the path he needs to take to complete the function or return to the original configuration.

Prompts can be used to solve this problem but "menus" provide a better solution.

As stated in Chapter II, menus provide an interface solution that most users recognize and feel comfortable with. Quick made some use of menus and they are a perfect tool to implement the AAFCE report functions and their multiple option paths.

After the choice of menus was made, another set of questions had to be answered. These questions were all design issues involving ergonomics and included:

1. What kind of menus?
2. How many menus?
3. How many options per menu?
4. How would the options be ordered?
5. How will the user invoke the menus and how will the options be selected?

There are three basic types of menus:

Static. These are considered ideal for novice users (23) because they are always on display and require no memorization.

Pop-up. These are activated under the cursor in special areas on the screen. They are invisible until activated and disappear once a selection has been made.

Pull-down. They are a hybrid of the other two. "This menu incorporates the visual properties of the state menu and the efficient display of the pop-up menu" (23:156). Each menu is given a name, which serves as a menu title. These titles are placed in a static menu on top of the screen. When the cursor is passed over the title, the menu, it represents, appears on the screen. If nothing is selected before the cursor leaves the menu area the menu disappears.

Quick's display keyboard is an example of a static menu. The rejection of this type has already been elaborated. Pop-up menus are better suited for experienced users as they require some memorization. Pull-down menus tend to frustrate users who are uncomfortable with a mouse.

A combination of static and pop-up menus was considered best suited for this application. The display keyboard provided a static display of the AAFCE menu. When

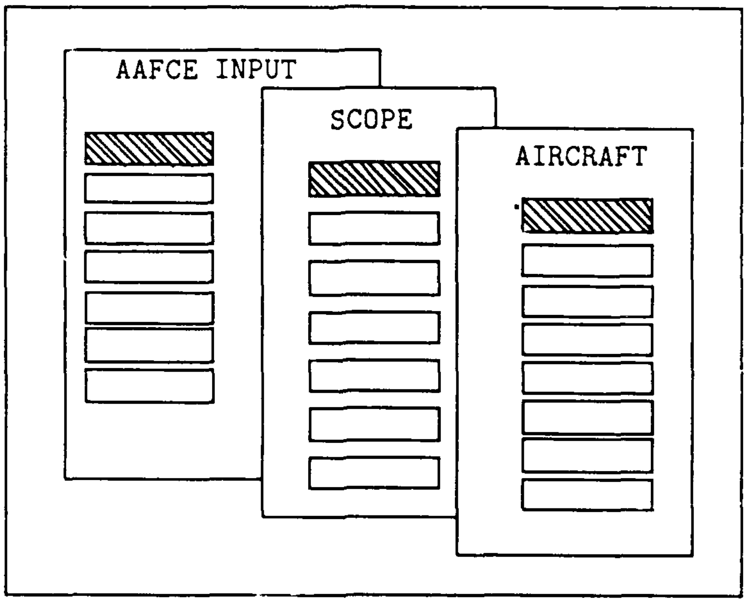


Figure 11. Overlapping Menus with Highlighting Options

this is selected the first option menu appears in the map area of the screen. Subsequent menus overlap the first. By leaving each menu visible and having them overlap, the user has a clear idea of where he is and what his next option is. Color is added to further aid the user. When the AAFCE function is invoked the button color on the display keyboard is changed to yellow (this is in keeping with Quick's original design) and it stays yellow until the function is exited. The first menu appears and when a report function is selected, the borders around it are changed to yellow and they stay that way until the option is complete. This colorization is used through every level and since the menus overlap so the invoked selections can be seen, the user can follow a highlighted path back to his original selection (see Figure 11).

The number of menus and the number of options per menu window were related issues - breadth versus depth. A main menu for the AAFCE functions is required and this menu would have the greatest breadth, 14 items (including 'exit'). This violated Hodgson's (10) recommendations, but adding a more general main menu appeared trivial. If the menu is readable, less depth and more breadth is desirable (25). The report functions require two more menu levels, but these levels present a logical division of options. The second level offered the scope of the reports and output device (AAFCE, ATAF, or airbase, and printer or screen). The last level, required by the sortie summaries, allows the user to select the type of aircraft.

The presentation of the options on the menus follows the example set by Kross in his input interface. This was not changed because there was no discernable logical order to either the main menu or aircraft menu. The scope was already ordered from larger to smaller. Letter codes for the functions were chosen for keyboard entry, because it is a one key selection process and alphabetic, to avoid confusion and ensure uniqueness. Numbering would have sufficed for the second and third level menus (less than ten options), but the first level would require a two key input for some options. Function keys are used in many applications but may not be familiar to new users.

The final issue concerning menus was how the user would control the display. The selection of a mouse, to invoke the functions on the display keyboard, was in keeping with Quick's design. We also decided that the actual clicking of the mouse button to display the menu gives the beginner a greater sense of control. Option selection in the menus can be made by keyboard or mouse. This gives the interface some latitude for user preference. Each menu is given a clearly defined exit option to always give the user a way to abandon the system's current state.

6.2.2 Buttons. The button is an icon that is easily recognizable by novice users and already employed by Quick in the interface. Like a physical button, it has to be 'pushed' to be activated. This is done by clicking the mouse over the button. The new interface requires additional buttons to exit active display and input functions, and to signal the end of list inputs. These buttons can be selected by the mouse, or the player can use the keyboard to enter the provided letter code, as the user preferred.

Buttons were also added to facilitate the screen report functions. Kross' report functions in the form-based input, currently allows one base's information to be displayed on the screen at a time. After the player reviews the information, he must clear the screen and enter a new base number to display another. Rather than force the user to invoke the function for every base of interest, the new interface allows the user to select multiple bases for display. After the user has selected the bases of interest (up to ten), buttons appear at the bottom of the map area identified by the base ids. This allows the user to display the data on a pop-up window in any order he wishes and redisplay bases as many times as he wants.

6.2.3 Display Forms. Another feature of the new interface is presenting the player a window for exhibiting base reports and accomplishing aircraft and logistics moves. When used for reports the window is a static display. When used for movement input, the window, or "form" is an interactive device which includes:

1. Necessary information required by the player to make command decisions.
2. "Spaces" to input the desired numbers of aircraft and/or logistics to move.
3. Readable instructions to guide the player through the process.

This form is a variation of a direct interface and represents the application functions of data retrieval from the TWX database, user updates, and the commitment of these updates back to the database.

6.3 User Input and Input Devices

The input functions require two basic operations, selecting the bases(s) to manipulate and entering the numerical data updates. After the user is done making his inputs, he must signal the computer that the data is ready to be processed. Base ids and aircraft and logistics moves are subjected to gross error checks by the system. The user is notified of errors and is asked to re-input his data. After the data is successfully input and the input involves a change to the database the user is then asked by the system if he is sure that he wants to make these moves. When the player answers 'yes' the changes are made to the TWX database.

Input methodology involves the interaction of the mouse and icons or keyboard and text. The decision was made to employ both methods, thereby giving the user a choice of using the one he is more comfortable with.

The new combined interface does not create any new icons but makes use of the existing ones that Quick designed. The interface allows the players to select base icons to represent the 'To' and 'From' bases for aircraft and logistics movement. The player is also allowed to make these selections by keyboard. Icons could have been used to represent aircraft and logistics for the players to manipulate, but there are drawbacks to this type of implementation:

1. The interface is intended for senior military officers, who may not be familiar with computer operations, but are familiar with filling out documents. Using icons, to represent the different aircraft and logistics, brings the interface to a level of simplicity that does not promote confidence in its use.
2. The icons for different aircraft and logistics could introduce a level of confusion that is avoided by their textual representations.
3. The icons have to represent predetermined numbers of aircraft and logistic amounts. This limits user flexibility or involves a clumsy and complicated method to cut up or combine icons to form different values.

Straight textual input for the numerical data was determined to be the best method. The user is presented with spaces to fill in the desired numbers and is allowed to edit them freely until he determines they are correct. Movement from one space to another is accomplished by using the mouse, arrow keys, or the enter/carriage return key. Before the user does jump to another space, he has to save the data in the current space. He must terminate that entry with the enter/carriage return. This will drop him to the next available space where he can move freely from then on. If the input data is not terminated in this manner it is lost to the system and must be reentered. This is a limitation to the HOOPS' string handling capabilities. A modification allowing you to specify the maximum number of characters would be useful.

User and system protection can be ensured through the HOOPS 'enable and selectability' functions. The enable functions allows the developer to select and restrict which input devices are active at any given time. Keyboard input can be explicitly restricted to selected keys and accepted by the system as a single character. Keys, which are not enabled, are ignored by HOOPS and are not placed in the event queue. This specific enabling function allows the user to make errors and gives him confidence that the errors will not affect the system.

The keyboard can also be enabled for string input and accepted as a string of characters, which have to be terminated by a enter/carriage return. Enabling string input involves more error checking by the system. String events include all printing keys on the keyboard. If the system is expecting numeric input, then a check is made to ensure that it is numeric. Other checks included verification that a base number is valid and numeric values fall within acceptable ranges.

The mouse is enabled as a selection device. Associated with this enabling is 'turning on' a graphic structure's selectability. When the blue player enters the AAFCE input function only the blue bases' selectability is turned on. If the player keys on blue units, red bases, or red units, the event is nct queued and the action is ignored by the system. Selection of options on the display keyboard is also limited. For example, not all of the display keyboard's functions are practical at every stage of user interaction. These functions are explicitly ignored in the code and no action taken when keyed upon, where they can cause disastrous results.

VII. Description of the Prototype

This chapter describes the existing prototype. It depicts the current configuration, sign-on procedures, and modifications to the existing system, and new extensions. The new display and input functions are summarized.

7.1 Configuration

The finished prototype runs under the UNIX operating system on a SUN 386i workstation. The prototype does not employ any DBMS facilities at this time. In order to test the interface's functionality, a set of test data files were designed. These data sets allow the interface to plot some base and unit markers, and base and unit symbols for each affiliation, but does not display boundaries, coastlines, and the forward edge of battle. They also contain information necessary to test the 'user friendliness' of the display and input functions. Once the ORACLE DBMS is in place, the appropriate SQL statements will be added and the geographical flat file will be transferred to the SUN. Then full integration and implementation testing will take place.

7.2 Sign-on Procedures

When the interface is started, it prompts the user for the name of the device which holds the game database and then the seminar number. These user responses are not used by the current prototype, but will be used in the future to connect to the appropriate database. There will be some modifications to this procedure because the database will be stored on the SUN workstation with the TWX system software. This configuration should require only the seminar number.

The user is then asked to input his affiliation (RED, BLUE, or WHITE) and his group's password. After the password is validated against the test password, the exercise proceeds. The operational system will validate the password and affiliation combination against the data in the seminar control table. If the password is incorrect, the user is given four more tries to reply correctly before the exercise is terminated. The purpose of the password system is to:

1. Ensure that the user is legitimate.
2. Filter information on opposing forces.
3. Load proper table names into the data retrieval statements.

After these initial prompts are answered, the interface checks the input status of the seminar. The system uses this information to display a status report which informs the players what input functions are be available to them. The information is also used to configure the display keyboard and control access to the AAFCE input functions. Red and Blue players will receive information pertaining to their affiliation. White users will be given the statuses of both groups. This status report includes:

Input Status. If the input portion of the system is locked then this will be the only message on the screen. The user will still be able to use the output display functions, but none of the input functions. Otherwise the user is informed that the system is open and further statuses are given.

AAFCE Status. These statuses inform the user which AAFCE input functions are still available. These functions are aircraft movement, aircraft rerole, air logistic's movement, and sortie calculations.

ATAF Statuses Though not implemented as part of this thesis effort the status of these functions are still displayed.

7.3 Modifications to the Output Display Interface

When possible the new system employed the original structures and functions created by Quick. These modifications were integrated into the code without changing the original program structure.

7.3.1 The Display Keyboard. The obvious differences in the new interface's initial display and Quick's interface are the context area and the display keyboard. Quick's interface had only the White team's display implemented; the new interface displays the actual player's affiliation. The new keyboard (see Figure 12) includes the AAFCE function

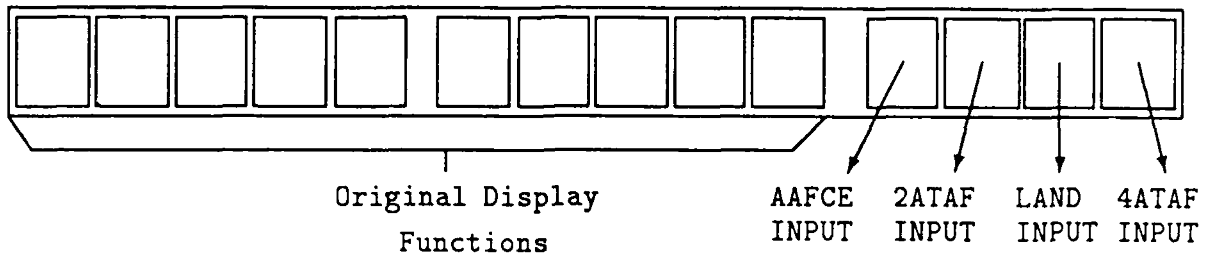


Figure 12. The Display Keyboard

and the Land, 2ATAF, and 4ATAF input functions, even though they are not implemented at this time. The display keyboard reflects the status report given earlier, if the system is locked then none of these functions will be displayed.

7.3.2 Help Function The 'Help' function is activated when the player enters the AAFCE function. When the 'Help' function is invoked, a display screen appears in the map area displaying information pertinent to where the player is in the AAFCE function. The display screen remains there until the player removes it by hitting any key. The 'Help' function can be activated at anytime during the AAFCE functions.

7.3.3 Show_Detail Function. Another modification was defaulting to the 'Show_Detail' when the player uses the mouse to key on a base or unit and the player is not using another function. The selectability of the base and unit markers was turned on in the new interface and the main input loop, 'Manipulate' was modified to accomplish this default. The 'Show_Detail' function, itself, was modified to limit the information retrieved for the detailed summary display. The players were still given unlimited access to information pertaining to their own bases and units, but the opposing team's information was filtered depending upon the intelligence level. The three levels of intelligence are suspected unit, unknown unit, and known unit (see Figure 13).

7.3.4 Unit Symbols. The intelligence data is also used to restrict the configuration of the unit symbols when the display is 'zoomed' to the third level of detail. If a opposition

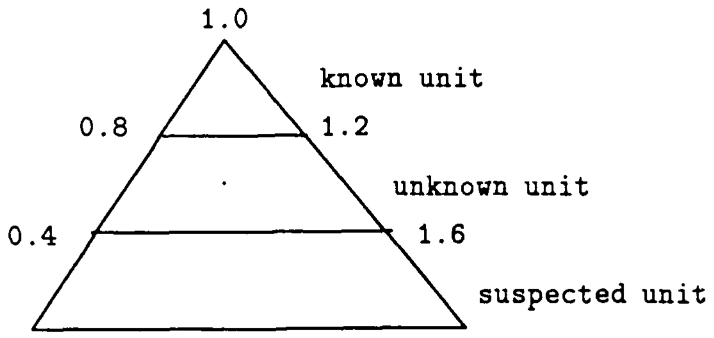


Figure 13. Intelligence Level

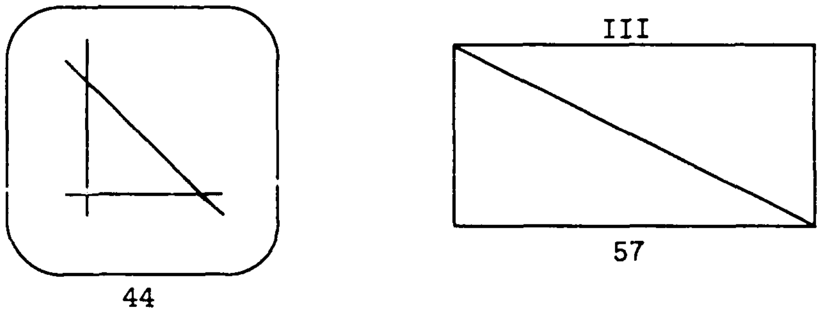


Figure 14. Base and Unit Symbols

unit's intelligence is in the lower level then just a box is displayed without the type, level, or ID. The ID was another modification to the unit and base symbols. The identification number was added below the symbol if permitted by the intelligence level or affiliation (see Figure 14).

The plotting scheme for unit symbols was changed to reflect the player's affiliation. Quick had the Blue units plotted first and then the Red units fully displayed in available spaces on the map. If there was a conflict in plotting the markers the Red symbols, the Blue unit was displayed fully with the Red unit contained within an overlap box. The modification plots the units first depending upon affiliation. If the player is Red then the Red units are plotted and then Blue units are filled in. The White users default to the original plotting scheme.

7.4 *New Extensions*

7.4.1 AAFCE Menu The AAFCE function is menu driven and uses three menus:

Main Menu This menu displays all the AAFCE options available to the user. All display options are displayed on the left of the menu and input functions on the right.

Scope Menu When a display option is selected, the user is allowed to limit the scope of his display from the whole AAFCE to individual bases. He is also allowed to direct his output when selecting individual bases to the printer or to the screen, other options are routed to the printer.

Aircraft Menu If the display option was the sortie summary, the user is given the option of selecting which type of aircraft to display.

When the menus are displayed all other interactive functions are suspended except the 'passive' functions like 'Help', 'Print_Screen', and 'Make_Suggest'. After an option is selected these screens disappear and the player is prompted through the option function. After the option is completed or exited, the main menu reappears for further processing. This menu will keep reappearing until the player chooses the 'exit' option.

7.4.2 Display Functions The display options, when selected will, display the scope menu. The scope menu options' implementations which are print directed have been delayed until the ORACLE DBMS is in place. The interface will then take advantage of ORACLE's report writing capabilities. The screen directed option is implemented using test data at this time.

After the scope selection is made the menus disappear with one exception — the sortie summary. The sortie summary will display another menu for aircraft selection. The user is then prompted to select which bases he wishes to display information on, by keying on the their markers or symbols with the mouse. When a base is selected a message is displayed in the message area informing the player of that fact. Blue unit, Red unit, and Red base selectability is turned off so the system will ignore selection of these structures.

The user is allowed to select up to ten bases for display at a time. Also, as an aid to the user, the 'Change_Vis', 'Pan', 'Zoom_in', and 'Zoom_Out' functions are activated.

There are three graphic structures of interest displayed on the screen during this selection process. In the right hand corner a counter is displayed that informs the user of the number of bases he has already selected. In the lower left hand corner are two buttons that lets the user control the function. The 'done' button is used to signal the system that the player is done inputting his list (this is not necessary if the player selects ten bases). The 'quit' button allows the player to exit the function at anytime. These buttons can be selected using the keyboard and the buttons first letter, or by using the mouse.

After the bases are selected, the counter disappears and ten more buttons appear above the 'done' and 'quit' buttons. These buttons contain the IDs of the bases selected or the words 'not used'. When the user selects one of ID buttons with the mouse a display screen appears in the middle of the map area containing the desired information. This display screen will be updated to show another base's information upon the selection of another ID button, or will disappear upon invocation of the 'done' or 'quit' buttons. The 'done' button takes on a new function in this phase of the option. When this button is selected the user is placed back into the base selection mode to select up to ten bases more.

7.4.3 Input Functions. The input functions do not have any other menus associated with them. When these functions are selected the main menu disappears and the user is prompted to select a base using the mouse or enter the base's ID in the message area. If the function requires two bases then the user will be prompted for each base separately. When the user uses the keyboard to enter the base IDs, he must terminate each entry with the enter/carriage control key and then the bases are validated. If the base does not exist, the user is alerted and he will be allowed to input again. As with the display base options, the 'Change_Vis', 'Pan', 'Zoom_in', and 'Zoom_Out' functions are enabled as an aid to the player in base selection.

After the base selection is completed the user is presented with an input form for making his decisions. These forms are basically the same as the display screen except there are 'spaces for the user to input his moves. The cursor is positioned in the first available space when the form first appears and can be moved to any other space with the mouse, arrow keys, or enter/carriage control key. Data entered in the spaces must be followed by the enter/carriage control key. After the input is saved, it is subjected to a gross error check. If an error exists the user is repositioned to input the data again. If the input is not terminated by the enter/carriage control key then the input data will not be saved.

The 'done' and 'quit' keys are also used during the input functions. The 'done' signals the system that the user is ready to have his inputs committed to the database. The system prompts the user if he is sure and if the user answers 'y' then the moves are committed. If the user answers 'n' the cursor is repositioned to the first space in the form to continue input. If the player does not change an input, it is still saved. After the changes are committed the form is erased and the user is prompted to enter another base. The 'quit' allows the user to exit the function completely and return to the AAFCE main menu.

VIII. Conclusion and Recommendations

This chapter reviews the steps taken in this project, the strengths and weaknesses of the current prototype, suggested enhancements, and conclusion.

8.1 Review of Steps

The purpose of this thesis effort was to combine the TWX graphic output display interface and the form-based input interface into one interactive interface. The new system had to maintain the functionality of the two current interfaces, it had to be easy to learn and use, and should not distract from the primary purpose of the exercise.

The first step was to review literature concerning ergonomics and interactive display programs. The purpose of this step was to identify keys to designing a 'good' interface. Next was an evaluation of graphical toolkits. The output display currently used the HOOPS toolkit and it was the choice for this effort. The HOOPS toolkit included high-level routines especially suited for cartographic displays and interactive programs.

The previous two steps were designed to gather background on interactive systems and the tools used in their development. The third step was the implementation of these ideas.

A design methodology was chosen and employed. The mthodology chosen was a combination of the Classic Life Cycle, Prototyping, and Iterative Design. The requirements gathering depended heavily on understanding the functions performed by the two current interfaces. After the initial requirements were determined, the material learned in the first two steps were used to design the new system.

After the design issues were settled, prototypes were developed; each implementing a specific objective in the requirements. Each prototype was tested for functionality against test data files. After the prototypes were successfully tested, they were merged together and subjected to integration testing.

The final prototype still requires work. The database retrieval statements have to be put into place and an objective evaluation has to be made by the users.

There were some problems encountered in this effort. More early interaction with the user should have been made to help determine system requirements. This is critical to any design of an interactive system and prototyping effort. Fortunately the HOOPS toolkit is easy to work with and modifications or additions should not be difficult to implement.

Another problem was the advertised portability of the HOOPS toolkit. The initial prototypes were developed on a PC and were to be transferred over to the SUN workstation for integration. There was no problem in the compilation of the interface's software in the new environment, but there were some unexpected effects resulting from some of the HOOPS commands. Commands such as 'Update_Display' and 'Bring_To_Front' were used liberally in the PC versions of these prototypes to get the desired display configuration. On the SUN workstation these commands apparently were not needed as frequently and their use caused an annoying flickering in the display. This called for some modification of the code in the new environment.

8.2 Strength sand Weaknesses of the New System

The obvious strength of the graphical player interface is that the TWX seminar can be run through one program. This interface is easy to learn and use which is a strength when used by the students. It is a weakness when used by the faculty. The faculty will tire quickly of the "hand-holding" by the interface and will want a more flexible system.

Another strength of the interface is the selection of HOOPS as a graphical toolkit. The HOOPS language is straightforward and object oriented, making the programming portion of this project relatively easy. HOOPS' style encourages the developer to build a library of graphical objects early in the program to be acted upon later as named entities. The creation of these objects was rather labor intensive and Quick's (18) idea of being able to create these objects through an interactive 'paint' program is interesting and deserves some future research.

Once the objects are built, HOOPS provides a wide variety of manipulation routines that can be used on them. Modifying specific objects has no effect on the other objects in the display. The inheritance feature allows groups of objects stored in a segment and to be affected together by single commands. There are also many high level routines available in HOOPS that were not available in other toolkits such as zooming and panning.

HOOPS was also an important tool in ensuring that the user and system were protected from each other. Since HOOPS allows the developer to specifically designate which input devices are active and which are ignored, there are no surprises when the user operates the wrong device. Events returned by this device are ignored by the system. Object selectability was another tool used for this protection. Items with their selectability turned off are ignored when selected by the user.

The portability of the system (not just HOOPS) is restricted to operating systems without a 64k segment limit. Portions of the interface code were too large for this limitation. This was avoided in the prototype design by limiting the size of the arrays and excluding functions that were not necessary to implement the specific objectives.

8.3 Enhancements

The implementation of the ORACLE SQL statements has to be completed before the system is operational. This should not be a problem because there is little difference between the ORACLE and INGRES SQL. One major difference that does exist is that ORACLE allows the use of variables to store table names. This enables the SQL 'from' statements to be loaded at run time after the team's affiliation is known. The primary advantage of using variables is the size of the SQL code can be cut in half. Currently the output display interface and the form-based interface repeat sections of SQL code, one section for Blue access and one section for Red access. One section can be eliminated with the use of variables.

Another modification that will be required for an operational system is the implementation of the ATAF and Land input functions. These functions should be possible to implement using the same methodology as the AAFCE input. Also the 'Help' function should be fully integrated into the system.

An expert user feature to the interface should be implemented. This feature should be delayed until Dallas Harken (9) has completed his Red player role automation effort.

The addition of a window containing all the team's base IDs and base names would be an interesting feature to add. This window could be brought up at any time and the user could mark bases in it, to be fed to the 'Show_Detail', the AAFCE report, and the AAFCE input functions.

Incorporating the HOOPS toolkit and the X Window System opens up many possible enhancements and should be considered for future research.

8.4 Conclusion

This thesis detailed the development of the graphical player interface. Additional benefits derived from this effort were insights into computer interaction and human factors, and the evaluation of graphic toolkits. The use of the HOOPS toolkit, even with the small problem stated earlier, should be strongly encouraged.

Bibliography

1. Abi-Ezzi, Salim S. and Albert J. Bunshaft. "An Implementer's View of PHIGS," *IEEE Computer Graphics and Application, 6*:12–21 (February 1986).
2. Air Force Wargaming Center, Maxwell AFB, Al. *Theater Warfare Exercise 1988*, 1987. unpublished manual.
3. Aronson, John. Director of Sales, Ithaca Software. telecon 13 Sep 1989.
4. Betts, Bill and others. "Goals and Objectives for User Interface Software," *Computer Graphics, 21*:73–78 (April 1987).
5. Brooks, Captain Michael D. *Developing a Database Management System and Air Simulation Software for the Theater War Exercise.* MS thesis, AFIT/GCS/ENG/87D-6, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Oh, December 1987 (ADA189681).
6. Enderle, G. and others. *Computer Graphics Programming.* Berlin, Ge.: Springer-Verlag, 1984.
7. Foley, J.D. and A. Van Dam. *Fundamentals of Interactive Computer Graphics.* Reading, Ma.: Addison-Wesley Publishing Company, 1972.
8. Grimes, Jack and others. "User Interface Design: Are Human Factors Principles Used?," *SIGCHI Bulletin, 17*:22–26 (January 1986).
9. Harken, Captain Dallas. *An Expert Automating Nuclear Strike Aircraft Replacement and Logistics Movement for TWX.* MS thesis, AFIT/GCS/ENG/89D-16, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Oh, December 1989.
10. Hodgson, Gordon M. and Stephen R. Ruth. "The Use of Menus in the Design of On-line Systems: A Retrospective View," *SIGCHI Bulletin, 17*:16–20 (July 1986).
11. Hudson, Scott E. "UIMS Support for Direct Manipulation Interfaces," *Computer Graphics, 21*:120–124 (April 1987).
12. Hutchins, Edwin L. and others. "Direct Manipulation Interfaces," *Human-Computer Interaction, 1*:311–338 (1985). cited by Captain Darrell Quick, Developing Map-Based Graphics for the Theater War Exercise (Masters Thesis), p.9.
13. Ithaca Software, Ithaca, NY. *HOOPS Graphics Software*, 1987.
14. Jones, Oliver. *Introduction to the X Window System.* Englewood Cliffs, NJ: Prentice-Hall, Inc, 1989.
15. Kross, Captain Mark S. *Developing New User Interfaces for the Theater War Exercise.* MS thesis, AFIT/GCS/ENG/87-19, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Oh, December 1987 (ADA189744).
16. Newman, William M. and Robert F. Sproull. *Principles of Interactive Graphics.* New York, New York: McGraw-Hill, Inc, 1979.

17. Pressman, Roger S. *Software Engineering, A Practitioners Approach.* New York, New York: McGraw-Hill, Inc, 1987.

18. Quick, Captain Darrell Anthony. *Developing Map-Based Graphics for the Theater War Exercise.* MS thesis, AFIT/GCS/ENG/88D-16, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Oh, December 1988.

19. Roth, Major Mark A. Associate Professor of Computer Systems. Personal interviews. Air Force Institute of Technology, Wright-Patterson AFB, OH, 1 August 1989 through 30 November 1989.

20. Saja, Allen D. "The Cognitive Model: An Approach to Designing the Human-Computer Interface," *SIGCHI Bulletin*, *16*:36–39 (July 1985).

21. Shneiderman, B. "The Future of Interactive Systems and the Emergence of Direct Manipulation," *Behavior and Information Technology*, *1*:237–256 (1982). cited by Scott E. Hudson in his article UIMS Support for Direct Manipulation Interfaces in the journal Computer Graphics (volume 21, April, 1987).

22. Shutz, JR, E. Eugene and Patrick S. Curran. "Menu Structure and Ordering of Menu Selections: Independent or Interactive Effects," *SIGCHI Bulletin*, *18*:79–81 (July 1985).

23. Sobelman, Gerald E. and David E. Krekelberg. *Advanced C: Techniques and Applications.* Indianapolis, In: Que Corporation, 1985.

24. Sommerville, Ian. *Software Engineering.* Wokingham, England: Addison-Wesley Publishing Co., 1989.

25. Tolle, John E. and others. "Effects of Variations in Menua Length and Number of Windows on User Search Time," *SIGCHI Bulletin*, *18*:73–74 (January 1987).

26. Van Deusen, Edmund. *Graphics Standards Handbook.* Laguna Beach, Ca: CC Exchange, 1985.

27. Wilcox, Captain Ken. *Extending the User Interface for the Theater War Exercise.* MS thesis, AFIT/GCS/ENG/88-19, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Oh, December 1988.

28. Yestingsmeier, Jan. "Human Factor Considerations in Development of Interactive Software," *SIGCHI Bulletin*, *16*:24–27 (July 1984).

Vita

PJ Gordon [redacted]

[redacted] After one year of college at Plymouth State University, New Hampshire, he enlisted in the Air Force. His first assignment was at OL-A 603 TCS, Reisenbach, Germany, working as a Radio Relay Repairman. In 1978 he was crosstrained to a computer programming specialist and was assigned to the Data Systems Design Center at Gunter AFS, Alabama. In 1982 PJ was given an early release from the Air Force to pursue a commission at the Alabama State University AFROTC detachment. In 1983 he received a bachelor's degree in Mathematics from Troy State University. He received a bachelor's degree in History from Auburn University and received his commission, from Alabama State, as a second lieutenant in the United States Air Force in 1984. His initial assignment at Offutt AFB, Nebraska was as an analyst/programmer for the Air Force Global Weather Central. He is currently a M.S. student in the School of Engineering at the Air Force Institute of Technology and a Captain in the United States Air Force. His next assignment is to the Air Force Communications Command at Scott AFB, Illinois.

[redacted]